Sommaire

1 INTRODUCTION 5
1.1 Les logiciels PICAXE 5
1.2 Etiquettes 7
1.3 Commentaires 8
1.4 Constantes 9
1.5 Symboles 10
1.6 Directives 11
1.7 Variables générales 14
1.8 Variables - Storage 16
1.9 Variables - Scratchpad 17
1.10 Variables - Système 18
1.11 Variables - fonctions spéciales 19
1.12 Variables - Opérations mathématiques 27
1.13 Variables - Opérateurs unaires 30
1.14 Conventions de nommage des ports d'entrée/Sortie 33

2 COMMANDES DU BASIC 34
2.1 adcconfig 34
2.2 adcsetup 36
2.3 backward 41
2.4 bcdtoascii 42
2.5 bintoascii 43
2.6 booti2c 44
2.7 branch 46
2.8 button 47
2.9 calibadc (calibadc10) 48
2.10 calibfreq 49
2.11 clearbit 50
2.12 compsetup 51
2.13 count 56
2.14 daclevel 59
2.15 dacsetup 60
2.16 debug 62
2.17 dec 63
2.18 disablebod 64
2.19 disabletime 65
2.20 disconnect 66
2.21 do ... loop 67
2.22 doze 68
2.23 eeprom (data) 69
2.24 enablebod 70
2.25 enabletime 71
2.26 end 72
2.27 exit 73
2.28 for...next 74
2.29 forward 75
2.30 fvrsetup 76
2.31 get 78
2.32 gosub (call) 79
2.33 goto 80
2.34 hi2cin 81

(c) Revolution Education Ltd.
All rights reserved

2.35	hi2cout	82
2.36	hi2csetup	83
2.37	hi2csetup - mode esclave (PICAXES X2 seulement)	84
2.38	hi2csetup - mode maître	85
2.39	halt	87
2.40	hibernate	88
2.41	high	89
2.42	high portc	90
2.43	hintsetup	91
2.44	hpwm	92
2.45	hpwmduty	95
2.46	hserin	96
2.47	hserout	98
2.48	hsersetup	99
2.49	hspiin (hshin)	101
2.50	hspiout (hshout)	102
2.51	hspisetup	103
2.52	i2cslave	106
2.53	if...then \elseif...then\else\endif	108
2.54	if...then {goto}	110
2.55	If...and/or..then {goto}	110
2.56	If porta ... then {goto}	111
2.57	If portc... then {goto}	111
2.58	If ... then exit	112
2.59	If ... and/or then exit	112
2.60	If ... then gosub	113
2.61	If ... and/or then gosub	113
2.62	inc	115
2.63	infrain	116
2.64	infrain2	118
2.65	infraout	119
2.66	input	122
2.67	inputtype	123
2.68	irin	126
2.69	irout	128
2.70	kbin	131
2.71	keyin	132
2.72	kbled (keyled)	134
2.73	let	135
2.74	let dirs / dirsc =	137
2.75	let dirsA / dirsB / dirsC / dirsD =	138
2.76	let pins / pinsc =	139
2.77	let pinsA / pinsB / pinsC / pinsD =	140
2.78	lookdown	141
2.79	lookup	142
2.80	low	143
2.81	low portc	144
2.82	nap	145
2.83	on...goto	146
2.84	On...gosub	147
2.85	output	148
2.86	owin	149
2.87	owout	150
2.88	pause	151
2.89	pauseus	152
2.90	peek	153

(c) Revolution Education Ltd.
All rights reserved

2.91	peeksfr	155
2.92	play	156
2.93	poke	157
2.94	pokesfr	159
2.95	pullup	160
2.96	pulsin	161
2.97	pulsout	162
2.98	put	163
2.99	pwm	164
2.100	pwmduty	165
2.101	pwmout	166
2.102	random	169
2.103	read	170
2.104	readadc	171
2.105	readadc10	172
2.106	readdac	173
2.107	readdac10	174
2.108	readi2c	175
2.109	readinternaltemp	176
2.110	readfirmware	177
2.111	readmem	178
2.112	readtable	179
2.113	readoutputs	180
2.114	readportc	181
2.115	readrevision	182
2.116	readsilicon	183
2.117	readtemp	184
2.118	readtemp12	186
2.119	readowclk	187
2.120	resetowclk	188
2.121	Readowsn	189
2.122	reconnect	191
2.123	reset	192
2.124	restart	193
2.125	resume	194
2.126	return	195
2.127	reverse	196
2.128	rfin	197
2.129	rfout	199
2.130	run	201
2.131	select case \ case \ else \ endselect	203
2.132	serin	205
2.133	serrxd	207
2.134	serout	208
2.135	sertxd	210
2.136	servo	211
2.137	servopos	213
2.138	setbit	214
2.139	setint	215
2.140	setintflags	218
2.141	setfreq	219
2.142	settimer	221
2.143	shiftin (spiin)	223
2.144	shiftout (spiout)	225
2.145	Sleep	227
2.146	sound	228

(c) Revolution Education Ltd.
All rights reserved

2.147 srlatch 229
2.148 srset / srreset 231
2.149 stop 232
2.150 suspend 233
2.151 swap 234
2.152 switch on/off 235
2.153 symbol 236
2.154 table 238
2.155 tablecopy 239
2.156 tmr3setup 240
2.157 toggle 242
2.158 togglebit 243
2.159 touch 244
2.160 touch16 245
2.161 tune 247
2.162 uniin 254
2.163 uniout 255
2.164 wait 257
2.165 write 258
2.166 writemem 259
2.167 writei2c 260

3 ANNEXES 261
3.1 Annexe 1 - Commandes 261
3.2 Annexe 2 - Autres mots-clés réservés 262
3.3 Annexe 3 : Labels réservés 263
3.4 Annexe 4 - Commandes potentiellement en conflit 264
3.5 Annexe 5 - variantes du X2 265
3.6 Annexe 6 - variantes du M2 266
3.7 Site WEB du fabricant: 267
3.8 Marque déposée : 268
3.9 Remerciements: 269

(c) Revolution Education Ltd.
All rights reserved

1 Introduction

4

Le manuel du PICAXE est divisé en 3 tomes :

Tome 1 - Premiers pas

Tome 2 - Les commandes du langage BASIC

Tome 3 - Circuits d'interfaçage avec les microcontrôleurs

Ce tome 2 fourni la syntaxe (avec des exemples détaillés) pour toutes les commandes BASIC acceptées par le système PICAXE. Il est destiné à servir de table de référence pour chaque commande du BASIC acceptée par le système PICAXE.

Dans la mesure où certaines commandes ne sont acceptées que par certains modèles de PICAXEs, un dessin en marge indique la taille des PICAXEs pour lesquels la commande est applicable. (NDT : voir cadre ci-dessous)

Quand on utilise la méthode de programmation graphique (Flowchart), seul un sous-ensemble de commandes est disponible par la simulation sur l'écran. Ces commandes sont indiquées par l'icone correspondante à coté de la description.

Pour plus d'informations d'ordre général et pour l'utilisation générale des PICAXEs, merci de vous référer au tome 1 : "Premiers pas"

Cette version en français est le résultat du travail collaboratif et désintéressé des membres du forum français PICAXE :

Besqueut, fuse, GAPAK, jojojo, MGU, PapyJP et PieM

Bien que ces personnes aient mis tout leur cœur et leurs compétences à réaliser un travail de qualité, des erreurs ou omissions ont pu se glisser dans la traduction. Ils ne sauraient en être tenus pour responsables, mais bien évidement seront heureux de prendre en compte les remarques des lecteurs. Dans certains cas des erreurs manifestes de la version anglaise ont même pu être corrigées.

Si vous ne pouvez assumer le risque lié aux erreurs de traduction, vous êtes priés de vous référer exclusivement à la documentation d'origine en anglais fournie par Revolution Education sur le site dont l'adresse figure en haut de chaque page.

Le numéro figurant en **blanc sur fond noir** en haut de chaque page correspond au numéro de page dans le manuel en anglais.

Contrairement à ce qui est écrit ci-dessus, il n'a pas été possible de reprendre les dessins en marge pour les PICAXEs concernés. A la place, une ligne spécifique apparait (ou apparaitra...) au début de la description de chaque commande.

Dans tout ce document le terme "PICAXE" doit être compris comme "microcontrôleur PICAXE" ou "puce PICAXE". Il existe des "cartes PICAXEs" et d'autres "composants PICAXE" qui sont décrits comme tels le cas échéant.

1.1 Les logiciels PICAXE

La principale application utilisée pour programmer les PICAXEs est dénommée "**PICAXE Programming Editor**" c'est à dire "Editeur de programmes PICAXE". Ce logiciel est libre de droits pour les utilisateurs de PICAXEs.

Merci de vous référer au tome 1 de ce manuel intitulé "Premiers Pas" pour les détails d'installation et un guide pas à pas. Merci de vous assurer que vous disposez de la dernière version, ce logiciel est en libre téléchargement depuis www.PICAXE.com

"AXEpad" est une version simplifiée **du "PICAXE Programming Editor"** utilisable sur LINUX et sur le système d'exploitation des MacIntosh. Il est compatible avec l'intégralité des commandes BASIC du présent manuel.

"Logicator for PIC micros" est une application de programmation graphique conçue pour un usage dans l'éducation.

Les programmes peuvent être développés à l'écran sous forme d'organigrammes. Ces organigrammes sont ensuite automatiquement convertis en langage BASIC et téléchargés vers les puces PICAXE.

"PICAXE VSM" est un simulateur au format Berkeley SPICE, capable de simuler intégralement un circuit électronique comportant des puces PICAXEs. Le programme BASIC peut être déroulé pas à pas tout en observant les réactions des entrées/sorties en interaction avec le programme.

La dernière version du programme est disponible sur le site web des PICAXEs à l'adresse www.PICAXE.com

Si vous avez une question au sujet d'une quelconque commande, merci de la poser sur le forum du site :

www.PICAXEforum.co.uk

(c) Revolution Education Ltd.
All rights reserved

1.2 Etiquettes

5

Les étiquettes (LABELS) sont utilisées comme repères tout au long d'un programme. Plus précisément, elles sont utilisées pour marquer une position dans un programme. Cette position peut être utilisée pour faire un "saut" au moyen d'une commande GOTO , GOSUB ou autre. Une étiquette peut être n'importe quel mot (à l'exception des mots réservés du langage BASIC) et peut contenir des chiffres et le caractère "_" (soulignement=trait du 8). Une étiquette doit commencer par une lettre ou le caractère soulignement (pas de chiffre au début donc), et doit être suivie immédiatement par le signe ":"

Le compilateur n'est pas sensible à la casse (les lettres majuscules et minuscules peuvent être utilisées indifféremment à tout moment).

Exemple :

```
main:
        high B.1          ; allume la sortie N°1
        pause 5000        ; attends 5 secondes
        low B.1           ; éteint la sortie N°1
        pause 5000        ; attends 5 seconds
        goto main         ; boucle vers l'étiquette de départ : main
```

Whitespace = Espace blanc

L'espace blanc est un terme utilisé par les programmeurs pour désigner la zone blanche située au début d'une ligne ou entre les lignes d'un programme pour améliorer la présentation sur les éditions. Ceci peut inclure des espaces, des tabulations ou des lignes vides.

Par convention, on placera toujours les étiquettes à l'extrême gauche d'une ligne. Toutes les autres commandes devraient être décalées d'une tabulation. Cette convention rend le programme plus agréable à lire et à suivre.

Newline = nouvelle ligne

Les commandes sont normalement placées sur des lignes séparées. Néanmoins, si vous le souhaitez, le caractère ":" peut être utilisé pour séparer différentes commandes sur une même ligne. Par exemple :

```
if pin1 = 1 then : high 1 : else : low 1 : endif
```

Line continuation= Prolongement de ligne

Si une ligne de programme est trop longue, il est possible de la poursuivre sur la ligne suivante en utilisant le caractère "_"

Par exemple :

```
        if pin1 = 1 then gosub _
                label1            ;suite de la ligne précédente
```

Code Collapsing = Masquage de code

Pour les programmes très longs, il est possible dans l'éditeur de programme de masquer des portions de code au moyen des caractère { et } afin de faciliter la lecture. Par exemple :

```
        {
                high 1
        }
```

1.3 Commentaires

5

Les commentaires sont utilisés pour ajouter de l'information dans un programme pour usage ultérieur. Ils sont totalement ignorés par le compilateur lors du téléchargement. Les commentaires peuvent commencer par une apostrophe ' ou un point-virgule ; et se poursuivent jusqu'à la fin de la ligne. Le mot clé REM peut également être utilisé pour débuter un commentaire.

Pour commenter plusieurs lignes, vous pouvez utiliser les directives #REM et #ENDREM

Exemples :

```
high 0          ; make output 0 high
low 0           REM make output 0 low

#rem    ; #rem out a number of lines
high 0
pause 2000
#endrem
```


(c) Revolution Education Ltd.
All rights reserved

1.4 Constantes

6

Les constantes sont des nombres "fixes" utilisés dans le corps du programme. Les PICAXEs utilisent des constantes de type word (tout nombre entre 0 et 65535 inclus)

Les constantes peuvent être déclarées de 4 façons différentes :

en décimal *(c'est à dire en base 10)*,

en hexadécimal *(c'est à dire en base 16),*

en binaire *(c'est à dire en base 2)*,

en ASCII *(c'est à dire en base 256).*

Base	***Explication***	***Exemples***
Décimal	Les nombre sont saisis directement sans aucun préfixe	`100 ; 100 décimal`

Hexadécimal	Les nombres sont précédés par le signe $ ou par 0x	`$64 ; 64 hex`
		`0x64 ; 64 hex`
		`B1 = B0 ^ $AA ; xor variable B0 with AA hex`

Binaire	Les nombres sont précédés par le signe %	`%01100100 ; 01100100 binary`
ASCII	Les chaines de caractères sont encadrés par des apostrophes "	`"Hello" ; "Hello" - equivalent à "H","e","l","l","o"`

(c) Revolution Education Ltd.
All rights reserved

1.5 Symboles

6

Les symboles peuvent être associés à des constantes, et peuvent êtres utilisés comme prête-nom pour toutes variables (voir plus loin le chapitre sur les Variables). Les valeurs constantes et les noms de variables sont associées à un symbole en faisant suivre le nom du symbole du signe = puis de la constante ou du nom de la variable. Les symboles peuvent être n'importe quel mot à l'exception des mots réservés du langage BASIC (par exemple SWITCH, STEP, OUTPUT, INPUT, etc... ne peuvent être utilisés comme symbole).

Les symboles peuvent comporter des chiffres et le signe soulignement "_" (tiret du 8). (flash, flash_1, etc...)

Le premier caractère ne peut pas être un chiffre (par exemple "1_flash" est interdit)

Les symboles peuvent être utilisés dans un programme à la place de la constante ou de la variable qu'ils représentent.

Pour plus de détails, reportez-vous à la commande SYMBOL plus loin dans ce manuel. L'utilisation de symboles ne change strictement rien à la place prise par le programme dans la mémoire du PICAXE.

Exemple :

```
symbol LED_ROUGE = B.7          ; défini un symbole pour une constante
symbol COMPTEUR = b0            ; défini un symbole pour la variable b0
COMPTEUR = 200                  ; Charge la variable b0 avec la valeur 200

BouclePrincipale:               ; défini une étiquette de programme
                                ; L'étiquette se termine immédiatement par le signe :
    high LED_ROUGE              ; Allume la sortie B.7
    pause COMPTEUR              ; attends 0,2 secondes
    low LED_ROUGE               ; Eteint la sortie B.7
    pause COMPTEUR              ; attends 0.2 secondes
    goto BouclePrincipale       ; boucle vers l'étiquette de départ : BouclePrincipale
```


(c) Revolution Education Ltd.
All rights reserved

1.6 Directives

7

Les "directives" sont utilisées par le logiciel pour Définitr le type de PICAXE utilisé et pour déterminer quelles sections du programme doivent être compilées. En conséquence, les directives ne font pas partie d'un programme PICAXE : ce sont des instructions à destination du "compilateur".

Toutes les directives commencent par le signe # et doivent être utilisées sur une seule ligne. Tout caractère imprévu placé après la directive est ignoré.

Les directives signalées "Seulement pour Programming Editor" fonctionnent uniquement avec le logiciel "Programming Editor" et ne seront pas reconnues par les autres logiciels.

#PICAXE xxx

Signale au compilateur quel PICAXE est utilisé. Cette directive Définit également automatiquement un label correspondant au type de PICAXE. Ainsi, #PICAXE 08m2 est aussi l'équivalent de #define 08m2. Si aucune directive #PICAXE n'est donnée, alors le système utilise par défaut le PICAXE choisi (voir le menu Affichage/Options/Mode dans Programming Editor)

Exemple : **#PICAXE 08m2**

#com device

Définit le port série/USB utilisé pour le téléchargement vers le PICAXE.

Examples:

```
#com 1     (Windows AXE026 serial)
#com 6     (Windows AXE027 USB*)
#com       /dev/ttyS0 (Linux AXE026 serial)
#com       /dev/ttyUSB0 (Linux AXE027 USB*)
#com       /dev/tty.usbserial-xxxx (Mac AXE027 USB*)
#com       1 (Windows CE AXE027 USB*)
#com /     dev/tty.iap (iPhone/iPod Touch AXE026 serial)
```

Veuillez noter que sur les systèmes LINUX, le nom du port COM est en réalité un de moins que le port COM. Ainsi, COM1 est "/dev/ttyS0". Sur les systèmes Max, xxxx est un numéro de série unique.

Le nom du périphérique est sensible à la casse. Vous devez écrire exactement comme ci-dessus.

**Reportez vous à la notice du câble USB AXE027 pour plus de précisions.*

#slot number

Sélectionne le logement du programme interne (0-3) ou le logement de programme i2c (4-7) sur les PICAXEs X2.

#revision number

Définit la version utilisateur du programme (1-254) sur les PICAXEs X2.

#no_data

Empêche le téléchargement des données EEPROM (seulement effectif pour les PICAXEs pour lesquels le programme et les données sont séparés)

#no_table

Empêche le téléchargement des données table ou EEPROM (seulement pour les PICAXEs X1 et X2) Cette directive active automatiquement la directive #no_data

(c) Revolution Education Ltd.
All rights reserved

#no_end

Ne pas ajouter automatiquement la commande "end" à la fin du programme.

#freq m4/m8/m16

Définit la vitesse par défaut de l'horloge système pour les PICAXEs 28X & 40X.

Inutile pour les autres PICAXEs qui utilisent automatiquement leur résonateur interne.

```
Exemple: #freq m8
```

#define label

Définit un label qui sera utilisé par les directives ifdef ou ifndef.

```
Exemple: #define clock8
```

Ne pas confondre la directive #define avec la commande symbol=

#define est une directive et , lorsqu'elle est utilisée avec #ifdef, Définit les portions de code qui seront compilées.

La commande "symbol" est utilisée dans un programme pour donner un nom plus parlant à une variable ou à une broche.

#undefine label

Retire un label de la liste courante des labels Définits.

```
Exemple: #undefine clock8
```

#ifdef /#ifndef label

#else

#endif

Compile le code d'un programme de façon conditionnelle suivant qu'un label est Définit(#ifdef) ou pas(#ifndef).

```
Exemple:    #define clock8
            #ifdef clock8
                  letb1=8       ' seul ce code sera compilé
            #else
                  letb1=4
            #endif
```

#error "commentaire"

Force une erreur de compilation à cet endroit.

```
Exemple:    #error        "Chantier en cours..."
```

#rem /#endrem

Met en commentaire plusieurs lignes de texte.

Exemple :

```
#rem
high 0
pause 1000
low 0
#endrem
```

#include "NomDeFichier"

Charge dans le programme en cours une portion de code enregistré dans le fichier "NomDeFichier".

```
Exemple : #include "c:\test.bas"
```

Remarque : nécessite la version 6 de l'éditeur de programme (PE6)

Directives "Seulement pour Programming Editor"

#simtask all/0/1/2/3 "Seulement pour Programming Editor"

Définit la tâche à lancer pendant la simulation en cas d'utilisation du multi-tâches sur un PICAXE M2.

Si aucune tâche n'est précisée, c'est la tâche 0 qui sera automatiquement utilisée.

Plusieurs tâches peuvent également être simulées simultanément en choisissant "all"

```
Exemples:   #simtask 1
            #simtask all
```

#sim axe 101/axe102/axe103/axe105/axe107/axe092 "Seulement pour Programming Editor"

Simule sur l'écran le kit de développement correspondant.

Exemple #sim axe105

#simspeed Valeur "Seulement pour Programming Editor"

Définit le délai (en millisecondes) entre chaque commande simulée.

#terminal off/300/600/1200/4800/9600/19200/38400 "Seulement pour Programming Editor"

Configure le "Terminal de port série" pour une ouverture automatique à la fin du téléchargement (à la vitesse indiquée).

```
Exemple: #terminal 4800
```

#gosubs 16/255 "Seulement pour Programming Editor"

Définit le mode de compilation des commandes "gosub" (16 ou 255) pour les anciens PICAXEs 18X et 28X.

```
exemple: #gosubs 16
```


(c) Revolution Education Ltd.
All rights reserved

1.7 Variables générales

10

La mémoire RAM est utilisée pour enregistrer les données temporaires dans des variables au fur et à mesure du déroulement du programme. Cette mémoire est effacée en cas de perte d'alimentation ou de réinitialisation du PICAXE. Il y a quatre types de variables RAM :

general purpose	Variables d'usage général
scratchpad	Zone de stockage temporaire (tampon circulaire)
storage	Espace additionnel de stockage et accès aux mémoires du PIC
special function	Fonctions spéciales du PICAXE

Référez vous à la commande "let" pour plus de détails sur les opérations mathématiques avec des variables.

Variables d'usage général

PICAXEs :	***Byte (Octets)***	***Variable bit***	***Variable byte***	***Variable word***
X2	56	bit0-31	b0-55	w0-27
X1	28	bit0-31	b0-27	w0-13
M2	28	bit0-31	b0-27	w0-13
Obsolètes	14	bit0-15	b0-13	w0-6

Il y a au moins 14 variables d'usage général de type Octet. Ces variables Byte ou Octet sont nommées de b0, b1, etc... Les variables Byte peuvent mémoriser un nombre entre 0 et 255 inclus. Les variables Byte ne peuvent pas contenir des nombres négatifs ou fractionnaires, et "déborderont" sans prévenir si vous dépassez les valeurs limites que sont 0 et 255. (Ainsi 254 + 3=1 de même que 2-3=255)

Néanmoins, pour les nombres plus grands, deux octets peuvent être combinés pour créer une variable "word", qui est capable d'enregistrer un nombre entier positif compris inclusivement entre 0 et 65535. Ces variables "word" sont nommées w0, w1, w2, etc... et sont construites de la façon suivante :

w0 = b1:b0
w1 = b3:b2
w2 = b5:b4
w3 = b7:b6
etc...

Ainsi, la partie la plus significative de w0 est b1, et la moins significative est b0.

De plus, il y a 32 variables de type bit (bit0, bit1, etc...); Ces variables bit peuvent être utilisées quand vous avez besoin d'un seul bit (0 ou 1) de capacité d'enregistrement. On parle également de variables booléennes (OUI ou NON) ou de drapeaux (actif ou inactif).

Les variables bit sont prises sur les premières variables octet :

b0 = bit7 : bit6 : bit5 : bit4 : bit3 : bit2 : bit1 : bit0

b1 = bit15 : bit14 : bit13 : bit12 : bit11: bit10: bit9: bit8

etc ...

Vous pouvez utiliser n'importe quelle variable word, byte ou bit dans toutes les expressions mathématiques et dans les commandes qui acceptent des variables. Néanmoins, prenez garde à ne pas réutiliser accidentellement la même variable byte ou bit que celle utilisée par ailleurs dans une variable word.

Accès indirect aux variables d'usage général (PICAXEs M2 et X2)

Sur ces PICAXEs il y a jusqu'à 256 variables d'usage général. Les premiers octets connus sous les noms b0, b1, b2, etc... peuvent être utilisés directement dans n'importe quelle commande (comme pour tous les autres PICAXEs). Les 256 octets (0-255) peuvent également être utilisés à la fois directement et indirectement.

Pour accéder directement à une valeur, les commandes peek (pour lire un octet) et poke (pour écrire un octet) peuvent être utilisées. Pour accéder indirectement à une valeur, il faut utiliser la variable virtuelle @bptr. @bptr est un nom de variable et peut être utilisé partout où une variable est acceptée (c'est à dire partout où vous pouvez écrire b1 par exemple). Néanmoins, la valeur de cette variable spéciale n'est pas son contenu direct (comme avec b1) mais son contenu indirect, à savoir l'octet pointé par la valeur de la variable bptr.

NDT : par exemple si bptr=12 alors @bptr=valeur contenue dans la variable N°12. C'est aussi b12.

Le compilateur accepte également les variables @bptrinc (post-incrémentation) et @bptrdec (post-décrémentation).

Chaque fois que la variable @bptrinc est utilisée, la valeur de bptr est automatiquement augmentée d'une unité (l'opération bptr=bptr+1 est automatiquement réalisée chaque fois que vous lisez ou que vous écrivez la valeur de @bptr). Ceci est idéal pour l'enregistrement de matrices à une seule dimension.

(c) Revolution Education Ltd.
All rights reserved

1.8 Variables - Storage

11

Les variables dites "storage" sont des emplacements mémoire utilisés pour enregistrer temporairement des octets de données. Ils ne peuvent être utilisées pour effectuer des opérations mathématiques, mais peuvent l'être pour enregistrer temporairement des octets de données au moyen des commandes **PEEK** et **POKE**.

La quantité d'emplacements de stockage disponible dépends du type de PICAXE utilisé La table suivante donne le nombre d'octets disponibles ainsi que leurs adresses :

Pour plus de détails, référez vous aux commandes PEEK et POKE.

PICAXE	*Quantité*	*Adresses décimales*	*Adresses Hexa*
08M2	99	28 à 127	$1C à $7F
18M2	227	28 à 255	$1C à $FF
18M2+, 14M2, 20M2	483	28 à 511	$1C à $1FF
28X2, 40X2	200	56 à 255	$38 à $FF
20X2	72	56 à 127	$38 à $7F
Tous les PICAXEs X1	95	80 à 126	$C0 à $EF

Les PICAXEs X1 et X2 disposent également d'une mémoire "ScratchPad" ; voir la page suivante.

PICAXEs obsolètes :

Tous les PICAXEs M	48	80 à 127	$50 à $7F
Tous les PICAXEs A	48	80 à 127	$50 à $7F
18X	96	80 à 127 192 à 239	$50 à $7F $C0 à $EF
28X, 40X	112	80 à 127 192 à 255	$50 à $7F $C0 à $FF
08	néant		

(c) Revolution Education Ltd.
All rights reserved

1.9 Variables - Scratchpad

12

Le "Scratchpad" est un espace de mémoire temporaire pour l'enregistrement de données matricielles.

En français ou utilise le terme "tampon de données" ou "tampon circulaire" en référence à la façon dont on accède aux données.

Les PICAXEs 28X1, 40X1 et 20X2 disposent de 128 octets de ScratchPad (0-127)

Les PICAXEs 28X2 et 40X2 disposent de 1024 octets de ScratchPad (0-1023)

Pour accéder directement à une donnée du ScratchPad, utilisez les commandes **GET** (lecture) et **PUT** (écriture).

Pour accéder indirectement aux données du ScratchPad, utilisez la variable virtuelle "@ptr"

@ptr est le nom d'une variable spéciale qui peut être utilisée dans n'importe quelle commande (c'est à dire à la place d'une variable b1). Néanmoins, la valeur de la variable n'est pas son contenu direct (comme dans le cas de la variable b1) mais la valeur de l'octet pointé par **ptr** dans le Scratchpad.

Par exemple : si ptr=33, alors @ptr est le contenu de la mémoire N°33 dans le ScratchPad.

Le compilateur accepte également les variables spéciales @ptrinc (post incrémentation) et @ptrdec (post décrémentation).

Chaque fois que ces variables sont utilisées dans une commande, la valeur du pointeur est automatiquement augmentée (respectivement diminuée) d'une unité. C'est à dire que ptr=ptr+1 juste après l'exécution de la commande.

Ceci est très commode pour enregistrer des matrices unidimensionnelles.

Par exemple :

```
ptr = 0                    ' Initialise le pointer à 0
serrxd @ptrinc,@ptrinc,@ptrinc,@ptrinc,@ptr
                           ' lit 5 octets sur le port série
ptr = 0                    ' Ré-initialise le pointer à 0
for b1 = 1 to 5
     sertxd (@ptrinc)      ' renvoie les 5 octets sur le port série
next b1
```

Reportez-vous aux commandes **PUT** et **GET** pour plus de détails.

(c) Revolution Education Ltd.
All rights reserved

1.10 Variables - Système

13

Les PICAXEs M2 disposent de 8 variables de type word réservées à l'usage du microprocesseur.

Néanmoins, si ces fonctionnalités matérielles ne sont pas utilisées, ces variables peuvent également être utilisées comme variables d'usage général.

Nom de la variable	*Variable associée*	*Explication*
s_w0	task	Le numéro de la tâche courante (en mode multitâches)
s_w1	-	Réservé pour usage ultérieur
s_w2	-	Réservé pour usage ultérieur
s_w3	-	Réservé pour usage ultérieur
s_w4	-	Réservé pour usage ultérieur
s_w5	-	Réservé pour usage ultérieur
s_w6	-	Réservé pour usage ultérieur
s_w7	time	Le temps écoulé (en secondes)

Les PICAXEs X1 et X2 disposent de 8 variables de type word et d'un octet de drapeaux réservés à l'usage du microprocesseur.

Nom de la variable	*Variable associée*	*Explication*
s_w0	-	Réservé pour usage ultérieur
s_w1	-	Réservé pour usage ultérieur
s_w2	adcsteup2	Pour les PICAXEs 28X2 : mot de poids fort du convertisseur ADC
s_w3	timer3	Pour les PICAXEs X2 : valeur du chronomètre N°3
s_w4	compvalue	Pour les PICAXEs X2 : résultat de la comparaison
s_w5	hserptr	Pointeur sur le tampon mémoire du port série physique.
s_w6	hi2clast	Dernier octet écrit sur le bus I2C physique (en mode esclave)
s_w7	timer	La valeur du chronomètre.

L'octet de drapeaux contient 8 valeurs booléennes :

Nom de la variable	*Variable associée*	*Explication*
flag0	hint0flag	Pour les PICAXEs X2 : interruption détectée sur B.0
flag1	hint1flag	Pour les PICAXEs X2 : interruption détectée sur B.1
flag2	hint2flag	Pour les PICAXEs X2 : interruption détectée sur B.2
flag3	hintflag	Pour les PICAXEs X2 : interruption détectée sur une des broches ci-dessus
flag4	compflag	Pour les PICAXEs X2 : bascule sur tout changement du comparateur
flag5	hserflag	Un octet a été reçu en tâche de fond sur le port série physique.
flag6	hi2cflag	Un octet a été transmis sur le bus I2C physique (en mode esclave)
flag7	toflag	Le chronomètre "Timer" a débordé.

(c) Revolution Education Ltd.
All rights reserved

1.11 Variables - fonctions spéciales

14

Les variables disponibles pour des fonctions spéciales dépendent du type de PICAXE :

PICAXE-08 / 08M / 08M2 : Registres à fonctions spéciales

pins=	Le port d'entrées/Sorties
dirs=	Le registre de direction (Définit quelles broches sont en entrées ou en sorties)
infra=	Synonyme de la variable b13, utilisées pour la commande infrain2 sur un 08M

Autres registres à fonctions spéciales uniquement disponibles sur 08M2

bptr	Le pointeur sur la mémoire RAM
@bptr	L'octet de la RAM pointé par bptr
@bptrinc	L'octet de la RAM pointé par bptr (avec post incrémentation)
@bptrdec	L'octet de la RAM pointé par bptr (avec post décrémentation)
time	La valeur courante du chronomètre (en secondes à 4MHz ou 16MHz)
task	La tâche courante

La variable **pins** est décomposée en variables booléennes individuelles ce qui permet de lire des entrées individuelles au moyen d'une commande **IF...THEN**. Seules les broches existantes ont une variable booléenne définite.

pins = x : x : x : pin4 : pin3 : pin2 : pin1 : x

De même, la variable **dirs** est décomposée en bits individuels. Seules les broches bi-directionnelles existantes sont supportées.

dirs = x : x : x : dir4 : x : dir2 : dir1 : x

PICAXE-14M2 / 18M2 / 20M2: Registres à fonctions spéciales

pinsB	Le port d'entrée B
outpinsB	Le port de sortie B
dirsB	Le registre de direction B
pinsC	Le port d'entrée C
outpinsC	Le port de sortie C
dirsC	Le registre de direction C
infra=	Synonyme de la variable b13, utilisées pour la commande infrain2 sur un 08M
bptr	Le pointeur sur la mémoire RAM
@bptr	L'octet de la RAM pointé par bptr
@bptrinc	L'octet de la RAM pointé par bptr (avec post incrémentation)
@bptrdec	L'octet de la RAM pointé par bptr (avec post décrémentation)
time	La valeur courante du chronomètre (en secondes à 4MHz ou 16MHz)
task	La tâche courante

Quand elle est utilisée à gauche d'une affectation, la variable pins s'applique aux broches de sortie. Ainsi :

```
let outpinsB = %11000000
```

définitera les sorties 7 & 6 au niveau haut et toutes les autres au niveau bas.

Quand elle est utilisée à droite d'une affectation, la variable pins s'applique aux broches d'entrée. Ainsi :

```
let b1 = pinsB
```

va charger la variable b1 avec l'état courant des broches du port B.

La variable **pinsX** est décomposée en variables booléennes individuelles ce qui permet de lire des entrées individuelles au moyen d'une commande **IF...THEN**. Seules les broches existantes ont une variable booléenne définite.

pinsB = pinB7 : pinB6 : pinB5 : pinB4 : pinB3 : pinB2 : pinB1 : pinB0

La variable **outpinsX** est décomposée en variables booléennes individuelles ce qui permet d'écrire directement sur les sorties. Seules les broches existantes ont une variable booléenne définite.

outpinsB = outpinB7 : outpinB6 : outpinB5 : outpinB4 : outpinB3 : outpinB2 : outpinB1 : outpinB0

De même, la variable **dirsX** est décomposée en bits individuels ce qui permet de définitr individuellement chaque broche comme une entrée ou une sortie. Seules les broches bi-directionnelles existantes sont supportées.
dirsB = dirsB7 : dirsB6 : dirsB5 : dirsB4 : dirsB3 : dirsB2 : dirsB1 : dirsB0

Reportez vous plus haut à la section "Variables générales" pour la description des variables @bptr, @bptrinc et @bptrdec.

PICAXE-14M/20M : Registres à fonctions spéciales (non applicable aux 14M2 / 20M2)

pins=	Le port d'entrée lorsqu'on lit le port
(out)pins=	Le port de sorties lorsqu'on écrit sur le port
infra=	variable utilisée pour la commande **infrain**
keyvalue=	Autre nom pour la commande **infra**, utilisée avec la commande **keyin**

Veuillez noter que la variable "pins" est une "pseudo" variable applicable aussi bien au port d'entrée qu'au port de sortie.

Quand elle est utilisée à gauche d'une affectation, la variable pins s'applique aux broches de sortie. Ainsi :

```
let pins = %11000000
```

Définiera les sorties 7 & 6 au niveau haut et toutes les autres au niveau bas.

Quand elle est utilisée à droite d'une affectation, la variable pins s'applique aux broches d'entrées. Ainsi :

```
let b1 = pins
```

va charger la variable b1 avec l'état courant des broches du port d'entrée.

De plus, veuillez noter que :

```
let pins = pins
```

signifie : fait en sorte que le port de sortie reflète l'état du port d'entrée.

Pour éviter toute confusion, il est recommandé d'utiliser la variable outpins dans ce dernier cas :

```
let outpins = pins
```

La variable **pins** est décomposée en variables booléennes individuelles ce qui permet de lire des entrées individuelles au moyen d'une commande **IF...THEN**. Seules les broches existantes ont une variable booléenne définite.

14M pins = x : x : x : pin4 : pin3 : pin2 : pin1 : pin0
20M pins = pin7 jusqu'à pin0

La variable **outpins** est décomposée en variables booléennes individuelles ce qui permet d'écrire directement sur les sorties. Seules les broches existantes ont une variable booléenne définite.

14M outpins = x : x : outpin5 : outpin4 : outpinx :out pin2 : outpin1 : outpin0
20M outpins = outpin7 jusqu'à outpin0

(c) Revolution Education Ltd.
All rights reserved

PICAXE-18 / 18A / 18M / 18X : Registres à fonctions spéciales (non applicable aux 18M2)

pins=	Le port d'entrée lorsqu'on lit le port
(out)pins=	Le port de sorties lorsqu'on écrit sur le port
infra=	variable utilisée pour la commande **infrain** (=b13 sur un 18M)
keyvalue=	Autre nom pour la commande **infra**, utilisée avec la commande **keyin**

Veuillez noter que la variable "pins" est une "pseudo" variable applicable aussi bien au port d'entrée qu'au port de sortie.

Quand elle est utilisée à gauche d'une affectation, la variable pins s'applique aux broches de sortie. Ainsi :

```
let pins = %11000000
```

définitera les sorties 7 & 6 au niveau haut et toutes les autres au niveau bas.

Quand elle est utilisée à droite d'une affectation, la variable pins s'applique aux broches d'entrée. Ainsi :

```
let b1 = pins
```

va charger la variable b1 avec l'état courant des broches du port d'entrée.

De plus, veuillez noter que :

```
let pins = pins
```

signifie : fait en sorte que le port de sortie reflète l'état du port d'entrée.

Pour éviter toute confusion, il est recommandé d'utiliser la variable outpins dans ce dernier cas :

```
let outpins = pins
```

La variable **pins** est décomposée en variables booléennes individuelles ce qui permet de lire des entrées individuelles au moyen d'une commande **IF...THEN**. Seules les broches existantes ont une variable booléenne définite.

pins = pin7 : pin6 : x : x : x : pin2 : pin1 : pin0

La variable **outpins** est décomposée en variables booléennes individuelles ce qui permet d'écrire directement sur les sorties. Seules les broches existantes ont une variable booléenne définite.

outpins = outpin7 : outpin6 : outpin5 : outpin4 : outpin3 : out pin2 : outpin1 : outpin0

(c) Revolution Education Ltd.
All rights reserved

PICAXE-28A / 28X / 40X: Registres à fonctions spéciales

pins=	Le port d'entrée lorsqu'on lit le port
(out)pins=	Le port de sorties lorsqu'on écrit sur le port
infra=	variable utilisée pour la commande **infrain** (=b13 sur un 18M)
keyvalue=	Autre nom pour la commande **infra**, utilisée avec la commande **keyin**

Veuillez noter que la variable "pins" est une "pseudo" variable applicable aussi bien au port d'entrée qu'au port de sortie.

Quand elle est utilisée à gauche d'une affectation, la variable pins s'applique aux broches de sortie. Ainsi :

```
let pins = %11000000
```

définitera les sorties 7 & 6 au niveau haut et toutes les autres au niveau bas.

Quand elle est utilisée à droite d'une affectation, la variable pins s'applique aux broches d'entrées. Ainsi :

```
let b1 = pins
```

va charger la variable b1 avec l'état courant des broches du port d'entrée.

De plus, veuillez noter que :

```
let pins = pins
```

signifie : fait en sorte que le port de sortie reflète l'état du port d'entrée.

Pour éviter toute confusion, il est recommandé d'utiliser la variable outpins dans ce dernier cas :

```
let outpins = pins
```

La variable **pins** est décomposée en variables booléennes individuelles ce qui permet de lire des entrées individuelles au moyen d'une commande **IF...THEN**. Seules les broches existantes ont une variable booléenne définite.

pins = pin7 : pin6 : pin5 : pin4 : pin3 : pin2 : pin1 : pin0

La variable **outpins** est décomposée en variables booléennes individuelles ce qui permet d'écrire directement sur les sorties. Seules les broches existantes ont une variable booléenne définite.

outpins = outpin7 : outpin6 : outpin5 : outpin4 : outpin3 : out pin2 : outpin1 : outpin0

PICAXE-28X1 / 40X1 : Registres à fonctions spéciales

pins=	Le port d'entrée lorsqu'on lit le port
outpins=	Le port de sorties lorsqu'on écrit sur le port
ptr	Le pointeur sur la mémoire tampon (scratchpad)
@ptr	L'octet de la mémoire tampon (scratchpad) pointé par ptr
@ptrinc	L'octet de la mémoire tampon (scratchpad) pointé par ptr (avec post incrémentation)
@ptrdec	L'octet de la mémoire tampon (scratchpad) pointé par ptr (avec post décrémentation)
flags	Les drapeaux du système

Quand elle est utilisée à gauche d'une affectation, la variable **outpins** s'applique aux broches de sortie. Ainsi :

```
let pins = %11000000
```

définitra les sorties 7 & 6 au niveau haut et toutes les autres au niveau bas.

Quand elle est utilisée à droite d'une affectation, la variable **pins** s'applique aux broches d'entrée. Ainsi :

```
let b1 = pins
```

va charger la variable b1 avec l'état courant des broches du port d'entrée.

La variable **pins** est décomposée en variables booléennes individuelles ce qui permet de lire des entrées individuelles au moyen d'une commande **IF...THEN**. Seules les broches existantes ont une variable booléenne définite.

pins = pin7 : pin6 : pin5 : pin4 : pin3 : pin2 : pin1 : pin0

La variable **outpins** est décomposée en variables booléennes individuelles ce qui permet d'écrire directement sur les sorties. Seules les broches existantes ont une variable booléenne définite.

outpins = outpin7 : outpin6 : outpin5 : outpin4 : outpin3 : out pin2 : outpin1 : outpin0

Le pointeur de mémoire tampon (scratchpad) est décomposée en variables booléennes individuelles :
ptr = ptr7 : ptr6 : ptr5 : ptr4 : ptr3 : ptr2 : ptr1 : ptr0

Reportez vous à la section "Variables : Scratchpad" pour plus de précisions sur les variables
@ptr, @ptrinc, @ptrdec

L'octet de drapeaux contient 8 valeurs booléennes :

Nom de la variable	*Variable associée*	*Explication*
flag0	-	Réservé pour usage ultérieur
flag1	-	Réservé pour usage ultérieur
flag2	-	Réservé pour usage ultérieur
flag3	-	Réservé pour usage ultérieur
flag4	-	Réservé pour usage ultérieur
flag5	hserflag	Un octet a été reçu en tâche de fond sur le port série physique.
flag6	hi2cflag	Un octet a été transmis sur le bus I2C physique (en mode esclave)
flag7	toflag	Le chronomètre "Timer" a débordé.

(c) Revolution Education Ltd.
All rights reserved

PICAXE-20X2 / 28X2 / 40X2 : Registres à fonctions spéciales

pinsA	Le port d'entrée A
dirsA	Le registre de direction A
pinsB	Le port d'entrée B
dirsB	Le registre de direction B
pinsC	Le port d'entrée C
dirsC	Le registre de direction C
pinsD	Le port d'entrée D
dirsD	Le registre de direction D
bptr	Le pointeur sur la mémoire RAM
@bptr	L'octet de la RAM pointé par bptr
@bptrinc	L'octet de la RAM pointé par bptr (avec post incrémentation)
@bptrdec	L'octet de la RAM pointé par bptr (avec post décrémentation)
ptr	Le pointeur sur la mémoire tampon (scratchpad)
@ptr	L'octet de la mémoire tampon (scratchpad) pointé par ptr
@ptrinc	L'octet de la mémoire tampon (scratchpad) pointé par ptr (avec post incrémentation)
@ptrdec	L'octet de la mémoire tampon (scratchpad) pointé par ptr (avec post décrémentation)
flags	Les drapeaux du système

Quand elle est utilisée à gauche d'une affectation, la variable pins s'applique aux broches de sortie. Ainsi :

```
let pinsB = %11000000
```

définitera les sorties 7 & 6 au niveau haut et toutes les autres au niveau bas.

Quand elle est utilisée à droite d'une affectation, la variable pins s'applique aux broches d'entrée. Ainsi :

```
let b1 = pinsB
```

va charger la variable b1 avec l'état courant des broches du port B.

La variable **pinsX** est décomposée en variables booléennes individuelles ce qui permet de lire des entrées individuelles au moyen d'une commande **IF...THEN**. Seules les broches existantes ont une variable booléenne définite.

pinsB = pinB7 : pinB6 : pinB5 : pinB4 : pinB3 : pinB2 : pinB1 : pinB0

De même, la variable **dirsX** est décomposée en bits individuels ce qui permet de Définitr individuellement chaque broche comme une entrée u une sortie. Seules les broches bi-directionnelles existantes sont supportées.
dirsB = dirsB7 : dirsB6 : dirsB5 : dirsB4 : dirsB3 : dirsB2 : dirsB1 : dirsB0

Le pointeur de mémoire tampon (scratchpad) est décomposée en variables booléennes individuelles :
ptr = ptr7 : ptr6 : ptr5 : ptr4 : ptr3 : ptr2 : ptr1 : ptr0

Reportez vous à la section "Variables : Scratchpad" pour plus de précisions sur les variables
@ptr, @ptrinc, @ptrdec

Reportez vous plus haut à la section "Variables générales" pour la description des variables @bptr, @bptrinc et @bptrdec.

(c) Revolution Education Ltd.
All rights reserved

L'octet de drapeaux contient 8 valeurs booléennes :

Nom de la variable	*Variable associée*	*Explication*
flag0	hint0flag	Interruption détectée sur la broche INT0
flag1	hint1flag	Interruption détectée sur la broche INT1
flag2	hint2flag	Interruption détectée sur la broche INT2
flag3	hintflag	Interruption détectée sur une des broches ci-dessus
flag4	compflag	Interruption liée à la bascule du comparateur
flag5	hserflag	Un octet a été reçu en tâche de fond sur le port série physique.
flag6	hi2cflag	Un octet a été transmis sur le bus I2C physique (en mode esclave)
flag7	toflag	Le chronomètre "Timer" a débordé.

Si une fonction matérielle n'est pas utilisée dans le programme, le développeur peut librement utiliser chaque drapeau comme une variable booléenne.

1.12 Variables - Opérations mathématiques

22

Les microcontrôleurs PICAXEs utilisent des calculs mathématiques sur 16 bits (word). Les entiers valides vont de 0 à 65535.

Tous les calculs internes se font sur 16 bits, même si par exemple la variable cible est une variable byte, soit un octet (8 bits) entre 0 et 255.

Si le résultat d'un calcul interne dépasse 255, un dépassement de capacité va intervenir sans être signalé.

Les calculs mathématiques sont effectués strictement de la gauche vers la droite. Contrairement aux autres ordinateurs et calculatrices, les PICAXEs ne donnent pas aux signes * et / une priorité supérieure aux signes + et -

En conséquence, l'opération 3+4x5 est calculée de la façon suivante :

3+4=7

7x5=35

Le microcontrôleur ne supporte ni les fractions, ni les nombres négatifs. Néanmoins, il est souvent possible de réécrire les formules de telle sorte que seuls des entiers soient utilisés, et non des nombres fractionnaires. Par exemple :
let w1 = w2 / 5.7
n'est pas valable, mais :
let w1 = w2 * 10 / 57

est mathématiquement équivalent et peut être calculé par un PICAXE.

Tous les PICAXEs supportent les fonctions suivantes :

opérateur	*alias*	*opération*	*précisions*
+		addition	
-		soustraction	
*		multiplication	mot de poids faible du résultat de la multiplication
**		multiplication	mot de poids fort du résultat de la multiplication
/		division	quotient de la division
//	%	modulo	reste de la division entière
MAX		résultat limité à une valeur maximale	
MIN		résultat limité à une valeur minimale	
AND	&	ET booléen	
OR	|	OU booléen	Alt Gr 6 sur un clavier français
XOR	^	OU EXclusif booléen	^ espace sur un clavier français
NAND		NON-ET booléen	
NOR		NON-OU booléen	
XNOR	^/	NON-OU EXclusif booléen	
ANDNOT	&/	ET-NON booléen	attention : ceci est différent de NAND
ORNOT	|/	OU-NON booléen	attention : ceci est différent de NOR

Les PICAXEs X1 et X2 supportent de plus :

opérateur	*alias*	*opération*	*précisions*
<<		décalage à gauche	Les bits sont décalés d'un rang vers la gauche
>>		décalage à droite	Les bits sont décalés d'un rang vers la droite
*/		multiplication	mot de poids intermédiaire du résultat de la

			multiplication
DIG		chiffre	retourne le chiffre correspondant à la position indiquée dans un nombre.
REV		permutation booléenne	permute le nombre de bits indiqué.

Les calculs mathématiques sont effectués strictement de la gauche vers la droite

Sur les PICAXEs X1 et X2, il est possible d'utiliser des parenthèses :
let w1 = w2 / (b5 + 2)

Sur tous les autres PICAXEs, les parenthèses ne sont pas autorisées :
let w1 = w2 / (b5 + 2)
n'est pas valable. Il faudra donc écrire :
let w1 = b5 + 2
let w1 = w2 / w1

Autres précisions :

Addition et Soustraction

Les opérateurs (+) addition et (-) soustraction fonctionnent comme ils sont censés le faire. Veuillez néanmoins prendre garde au risque de dépassement de capacité sans aucun avertissement si les valeurs minimales ou maximales possibles pour une variable (0-255 pour un octet, 0-65535 pour une variable word) sont dépassées.

Multiplication et Division

Lorsque vous multipliez deux nombre de 16 bits (variable word), le résultat est un nombre de 32 bits. (deux words)

L'opérateur de multiplication (*) donne les 16 bits de poids faible de la multiplication word*word

L'opérateur (**) donne les 16 bits de poids fort de la multiplication word*word.

Sur certains PICAXEs, l'opérateur (*/) donne les 16 bits de poids intermédiaire.

Dans ces condition, l'opération mathématique standard : **$eeffgghh = $aabb x $ccdd**

s'écrit sur un PICAXE :
$gghh=$aabb * $ccdd
$eeff =$aabb ** $ccdd

Sur un X1 ou un X2, vous pouvez également obtenir les 16 bits intermédiaires :
$ffgg =$aabb */ $ccdd

L'opérateur (/) division donne le quotient de la division d'un word par un autre word.
L'opérateur (// ou %) modulo donne le reste de la division entière de ces mêmes words.

Max and Min

L'opérateur MAX permet de limiter une valeur à un plafond préDéfinit. Dans l'exemple suivant, la valeur ne dépassera jamais 50. Si le résultat d'un calcul dépasse 50, alors l'opérateur MAX limitera le résultat à au plus 50.

```
let b1 = b2 * 10 MAX 50
```

```
if b2 = 3 then b1 = 30
if b2 = 4 then b1 = 40
if b2 = 5 then b1 = 50
if b2 = 6 then b1 = 50   ' limited to 50
```

De même l'opérateur MIN permet de limiter une valeur à un seuil préDéfinit. Dans l'exemple suivant, la valeur ne sera jamais inférieure à 50. Si le résultat d'un calcul passe en dessous de 50, alors l'opérateur MIN limitera le résultat à au moins 50.

```
let b1 = 100 / b2 MIN 50
```

```
if b2 = 1 then b1 = 100
if b2 = 2 then b1 = 50
if b2 = 3 then b1 = 50 ' limited to 50
```

AND, OR, XOR, NAND, NOR, XNOR, ANDNOT, ORNOT
Les opérateurs AND, OR, XOR, NAND, NOR, XNOR fonctionnent sur chaque bit d'une variable.
A ANDNOT B signifie par exemple "A et l'inverse de B", A et B étant des variables booléennes ou les bits individuels d'une variable.

On utilise souvent la commande AND (&) pour masquer certains bits d'une variable.
Par exemple, si on souhaite ne prendre en compte que les bits N°1 et N°2 d'un port d'entrée, on peut utiliser un masque :

```
let b1 = pins & %00000110
```

b1 ne sera modifié que si les broches N°1 ou N°2 changent.

<< , >>

Ces opérateurs réalisent N décalages à gauche ou à droite des bits d'une variable. C'est exactement la même chose que de multiplier ou de diviser N fois cette variable par 2. Notez que les bits qui sortent (à gauche ou à droite) des 16 bits d'un word sont perdus. Notez également que les bits qui entrent (à droite ou à gauche) sont à zéro.

```
let b1 = %00000110 << 2
b1 vaut maintenant %00011000
```

DIG

L'opérateur DIG donne le nombre décimal correspondant au chiffre situé à la position (0-4, de la droite vers la gauche) d'un nombre de 16 bits. Ainsi le chiffre N°0 de 27890 est 0 et le chiffre N°3 est 7.

Pour récupérer la valeur ASCII de ce nombre, il suffit d'ajouter "0" (c'est à dire 32).

```
let b1 = b2 DIG 0 + "0"
```

Voyez également les commandes BINTOASCII et BCDTOASCII.

REV

L'opérateur REV retourne N bits dans un nombre de 16 bits.

Ainsi pour retourner les 8 bits de %10110000, il faut écrire :

```
let b1 = %10110000 REV 8
b1 vaut maintenant  %00001101
```

1.13 Variables - Opérateurs unaires

25

Opérateurs unaires disponibles sur tous les PICAXEs :

opérateur	*exemple*	*précisions*
NOT	let b1 = NOT pins	Les bits à zéro passent à un et réciproquement
-	let b1 = -b1	Evidement, il y a dépassement de capacité vers le bas : donc le PICAXE ajoute 256 pour une variable byte ou 65536 pour une variable word

Les PICAXEs X1 et X2 disposent de plus des opérateurs unaires suivants :

opérateur	*précisions*
SIN	100 fois le sinus d'un angle exprimé en degrés (entre 0 et 65535)
COS	100 fois le cosinus d'un angle exprimé en degrés
SQR	racine carrée
INV	synonyme de NOT
NCD	encodeur en base 2
DCD	décodeur de base 2
BINTOBCD	converti une valeur binaire en BCD
BCDTOBIN	décode une valeur BCD vers un nombre binaire

Les PICAXEs X2 disposent de plus des opérateurs unaires suivants :

NOB	compte les bits à 1
ATAN	donne l'arc tangente d'une valeur (entre 0 et 45 degrés)

Les opérateurs unaires doivent être la première commande d'une ligne. Néanmoins, ils peuvent être suivis d'autant d'opérations que nécessaire. Par exemple :

```
let b1 = sin 30 + 5  ' est valide

let b1 = 5 + sin 30  ' donne une erreur à la compilation
```

Informations complémentaires :

NOT

L'opérateur unaire NOT fait le complément à 2 de chaque bit d'une variable.

```
let b1 = NOT %01110000

b1 vaut maintenant %10001111
```

SIN and COS

L'opérateur unaire **SIN** donne un nombre proportionnel au sinus d'une valeur exprimée en degrés. Le système utilise une table de 45 valeurs discrètes dans chaque quadrant, ce qui donne un résultat rapide et relativement fiable.

L'opérateur SIN fonctionne seulement pour des nombres entiers positifs. Néanmoins, il suffit d'ajouter un tour, soit 360 degrés si vous disposez d'une valeur négative. Ainsi sin(-30°) = sin(360° -30°) = sin (330°)

Dans la mesure où un sinus est toujours compris entre -1 et +1 et que les PICAXEs ne manipulent que des nombres entiers, le résultat fourni est en fait 100 fois le sinus, ce qui augmente quelque peu la précision du résultat.

Ainsi sin(30) = 0.5, mais le PICAXE donnera 50

Les nombres négatifs sont signalés en positionnant à 1 le bit N°7 du résultat.

Ceci a pour effet que les nombres négatifs apparaissent comme si on leur avait ajouté 128.

Ainsi :
b1 = sin 210

b1 vaut maintenant 128+50 soit 178

L'opérateur unaire **COS** fonctionne de façon tout à fait similaire.
SQR

L'opérateur unaire "racine carrée" donne la valeur entière de la racine carrée, à partir de 10 itérations de la formule N-R en utilisant une racine égale à la moitié de la valeur. Cette méthode donne rapidement un résultat précis. Notez que dans la mesure où les PICAXEs n'utilisent que des nombres entiers, le résultat sera arrondi à l'entier inférieur.

NDT : dans certains cas, vous pouvez multiplier la valeur par 100 pour disposer d'un peu plus de précision

let b1=	sqr 64	sqr 63	sqr 6300
b1 vaut maintenant	8	7	79
valeur exacte	8	7,93725393319	79,3725393319

INV (~)
Synonyme de NOT

NCD

L'opérateur d'encodage donne le rang du dernier bit de la valeur positionné à 1 en partant de la droite. En conséquence, le résultat est normalement compris entre 1 et 16, sauf si la valeur vaut zéro auquel cas, le résultat est également zéro.

let b1=	ncd %00000100	ncd 0	%1000000010000000
b1 vaut maintenant	3	0	16

DCD

L'opérateur de décodage prend une valeur comprise entre 0 et 15 et fournit un nombre dont le bit de rang donné est positionné à 1.

let b1=	dcd 3	dcd 8	let w1=dcd 15
b1 vaut maintenant	%00001000	%100000000	w1 vaut %1000000000000000

*NDT : notez que les opérateurs NCD et DCD **ne sont pas** symétriques...*

BINTOBCD

NDT : la notation BCD permet d'afficher un nombre en hexadécimal comme si c'était un nombre décimal...

L'opérateur unaire **BINTOBCD** convertit une valeur en un nombre codé BCD. Veuillez noter que la plus grande valeur qui puisse ainsi être codée dans un octet est 99 et 9999 pour un word.

let b1=	bintobcd 99	let w1=	bintobcd 9999
b1 vaut maintenant	$99=153	w1 vaut maintenant	$9999=39321

BCDTOBIN

L'opérateur unaire **BCDTOBIN** fait l'inverse de **BINTOBCD**
let b1 = bcdtobin $99
let b1 = bcdtobin 153

b1 vaut maintenant 99

(c) Revolution Education Ltd.
All rights reserved

NOB (X2 seulement)

L'opérateur unaire **NOB** compte le nombre de bits positionnés à 1.

```
let b1 = NOB %10100111
```

```
b1 vaut maintenant 5
```

ATAN (X2 seulement)

L'opérateur unaire **ATAN** donne l'arc-tangente pour des angles compris entre 0 et 45 degrés. Ceci peut servir par exemple pour calculer la direction d'un robot. Dans la mesure où la tangente d'un angle compris entre 0 et 45 degrés est comprise entre 0 et 1 et que les PICAXEs ne manipulent que des nombres entiers, cette valeur doit être multipliée par 100 pour que le résultat ait une précision acceptable.

let b1=	atan 100	atan 10	atan 1
b1 vaut maintenant	45	5	0
valeur exacte	45°	5,710°	0,573°

(c) Revolution Education Ltd.
All rights reserved

1.14 Conventions de nommage des ports d'entrée/Sortie

27

Les premières puces PICAXEs ne disposaient que d'un maximum de 8 entrées et de 8 sorties. En conséquence, il n'était nullement nécessaire de disposer d'une convention de nommage puisqu'il n'existait qu'un seul port d'entrée et un seul port de sortie. Ainsi, les broches d'entrée et de sortie étaient simplement désignées par leur numéro, par exemple :

Commandes en sortie	*Commandes en entrée*
high 1	count 2, 100, w1
sound 2, (50,50)	pulsin 1, 1, w1
serout 3, N2400, (b1)	serin 0, N2400, b3

Toutefois, sur les PICAXEs M2 et X2 plus récents, une grande souplesse est disponible, autorisant pratiquement chaque broche à être configurée en tant qu'entrée ou en tant que sortie. Ceci donne évidement plus de 8 entrées possibles et plus de 8 sorties également. Une convention de nommage est donc devenue nécessaire. En conséquence, les broches d'un PICAXE sont désormais désignées par la notation PORT.PIN (où PIN représente la broche du PICAXE) Jusqu'à 4 ports (A,B,C,D) peuvent être disponibles suivant le nombre de broches du PICAXE considéré.

Commandes en sortie	*Commandes en entrée*
high B.1	count A.2, 100, w1
sound C.2, (50,50)	pulsin B.1, 1, w1
serout A.3, N2400, (b1)	serin C.0, N2400, b3

Pour la commande **if... then,** dans les expressions qui testent l'état d'une broche, la convention de nommage des variables représentant ces broches a évolué de façon similaire :

if pin1 =1 then...	devient	if pinC.1 = 1 then...

Les noms des variables associées aux ports évoluent de façon similaire :

broches en entrée	pins	devient	pinsA, pinsB, pinsC, pinsD
broches en sortie	outpins	devient	outpinsA, outpinsB, outpinsC, outpinsD
direction des broches	dirs	devient	dirsA, dirsB, dirsC, dirsD

Ce manuel utilise généralement la nouvelle convention PORT.PIN dans les exemples, sauf si l'exemple est spécialement décrit pour un PICAXE obsolète.

Merci de vous référer au schéma de brochage (dans le Tome 1 du présent manuel) correspondant au PICAXE que vous utilisez.

Faites attention au fait que le N° de broche d'entrée/sortie ne correspond absolument pas au numéro d'ordre de la patte de la puce !

(c) Revolution Education Ltd.
All rights reserved

2 Commandes du BASIC

2.1 adcconfig

28 (tous les PICAXEs M2, 28X2 et 40X2)

Syntaxe:

adcconfig config

- config est une constante/variable spécifiant la configuration de ADC.

Fonction:
Configure la référence de tension de la conversion analogique /numérique (ADC.)

Information:

La valeur par défaut de Vref+ pour la conversion ADC est la tension d'alimentation (V+) et la valeur par défaut Vref- est 0V, donc la gamme de tension mesurée est la même que la tension d'alimentation du PICAXE.
Toutefois, si nécessaire, les tensions Vref peuvent être modifiées par une broche externe en utilisant l'instruction **adcconfig**

PICAXE X2

Bit 3,2 = 11 ne pas utiliser
= 10 VRef+ est FVR (voir la commande FVRsetup)
= 01 VRef+ est une broche externe
= 00 VRef+ est V+ (tension d'alimentation du PICAXE)
Bit 1,0 = 11 ne pas utiliser
= 10 ne pas utiliser
= 01 VRef- est une broche externe
= 00 VRef- est 0V

PICAXE M2

Bit 2 = 1 VRef- est une broche externe (si présente)
= 0 VRef- **est** 0V
Bit 1,0 = 11 VRef+ est FVR (Voir FVR)
= 10 VRef+ est une broche externe (si présente)
= 01 ne pas utiliser
= 00 VRef+ est V+ (tension d'alimentation du PICAXE)

PICAXE	Broche externe Vref+	Broche externe Vref-
08M2	C.1	n/a
14M2	B.1	n/a
18M2	n/a	C.2
20M2	B.0	n/a
28X2	A.3	A.2
40X2	A.3	A.2

Exemple (18M2):

```
fvrsetup FVR2048          ; configure la tension de référence interne FVR à 2.048V
```


(c) Revolution Education Ltd.
All rights reserved

```
adcconfig %011          ; configure Vref+ comme étant FVR (donc 2.048V) car bits 1 et 0 sont à
                        1, et Vref- à 0V, car bit 2 est à 0
```


(c) Revolution Education Ltd.
All rights reserved

2.2 adcsetup

29 (tous les PICAXEs M2, 20X2, 28X2 et 40X2)

Syntaxe :

{let} adcsetup = channels

"channels" est une valeur numérique ou un masque désignant le ou les ADC à mettre en œuvre.

Concerne :

voir ci-dessous

Commentaire :

1/ L' utilisation de cette instruction diffère suivant le modèle de PICAXE utilisé. Se reporter à une des sections ci-dessous.

Section 1 :

PICAXE-28X2 (PIC18F25K22)

PICAXE-40X2 (PIC18F45K22)

Section 2

PICAXE-28X2-5V (PIC18F2520)

PICAXE-40X2-5V (PIC18F4520)

Section 3

PICAXE-28X2-3V (PIC18F25K20)

PICAXE-40X2-3V (PIC18F45K20)

PICAXE-20X2 (PIC18F14K22)

Section 4

Tous les modèles M2 (08M2, 14M2, 18M2, 20M2)

2/ Sur les modèles X2, il est nécessaire de configurer les broches utilisées avec l' instruction "readadc" ou "readadc10"

Pour tous les autres modèles, cette configuration est automatique.

.

3/ sur les modèles M2, adcsetup est automatiquement mise à jour dès lors que l'on utilise une instruction "readadc", "readadc10 " ou "touch". Par conséquent cette instruction n'est utile que si l'on veut forcer une entrée en entrée digitale en forçant le bit correspondant en 0.

Section 1

PICAXE-28X2 (PIC18F25K22) (Pas les versions -5V or -3V)

PICAXE-40X2 (PIC18F45K22) (Pas les versions -5V or -3V)

(c) Revolution Education Ltd.
All rights reserved

adcsetup est un masque

Avec ces modèles, le choix individuel des broches à utiliser comme entrée adc est possible.
Il suffit de placer le bit correspondant du masque en 1.

Remarque :

Le bit correspondant est automatiquement placé en 1 si l' on utilise une instruction read , readadc10, touch ou touch16

variable adcsetup

Bit 0 - ADC0	Bit 8 - ADC8
Bit 1 - ADC1	Bit 9 - ADC9
Bit 2 - ADC2	Bit 10 - ADC10
Bit 3 - ADC3	Bit 11 - ADC11
Bit 4 - ADC4	Bit 12 - ADC12
Bit 5 - ADC5	Bit 13 - ADC13
Bit 6 - ADC6	Bit 14 - ADC14
Bit 7 - ADC7	Bit 15 – non utilisé

variable adcsetup2

Bit 0 – ADC16	Bit 8 - ADC24
Bit 1 - ADC17	Bit 9 - ADC25
Bit 2 – ADC18	Bit 10 - ADC26
Bit 3 - ADC19	Bit 11 - ADC27
Bit 4 - ADC20	Bit 12 - non utilisé
Bit 5 - ADC21	Bit 13 - non utilisé
Bit 6 - ADC22	Bit 14 - non utilisé
Bit 7 - ADC23	Bit 15 - non utilisé

Tensions de référence :

Par défaut, la tension maximum Vref+ est la tension d'alimentation (V+) et la tension minimum est 0V.
Donc la plage de conversion est égale à la tension d'alimentation du PICAXE.

Néanmoins ces tensions de référence peuvent être modifiées en utilisant l'instruction " adcconfig"

Exemple:

```
let adcsetup = %0000000000001111 ; ADC 0,1,2,3 en service
```

Section 2

PICAXE-28X2 -5V (PIC18F2520)

PICAXE-40X2 -5V (PIC18F4520)

adcsetup est une valeur numérique (channels)

(c) Revolution Education Ltd.
All rights reserved

Avec ces modèles,le choix individuel des broches à utiliser en entrée adc n' est pas possible.

S' assurer d'abord que la broche est configurable en entrée adc et qu' elle est configurée en entrée.

On ne peut configurer le ni(ème) ADC(n) en entrée analogique que si les (n-1) adc le sont.

Par exemple on ne peut choisir ADC3 que si DC0-2 sont configurés en entrée analogique.

Channels (colonne 1)

0 none none

1 ADC0 ADC0

2 ADC0,1 ADC0,1

3 ADC0,1,2 ADC0,1,2

4 ADC0,1,2,3 ADC0,1,2,3

5 ADC0,1,2,3,8 ADC0,1,2,3,5

6 ADC0,1,2,3,8,9 ADC0,1,2,3,5,6

7 ADC0,1,2,3,8,9,10 ADC0,1,2,3,5,6,7

8 ADC0,1,2,3,8,9,10,11 ADC0,1,2,3,5,6,7,8

9 ADC0,1,2,3,8,9,10,11,12 ADC0,1,2,3,5,6,7,8,9

10 - ADC0,1,2,3,5,6,7,8,9,10

11 - ADC0,1,2,3,5,6,7,8,9,10,11

12 – ADC0,1,2,3,5,6,7,8,9,10,11,12

ADC4,5,6,7 n' existent pas sur le modèle 28X2-5V .

ADC4 n' existe pas sur le modèle 40X2-5V .

Tension de référence :

Par défaut, la tension maximum Vref+ est la tension d' alimentation (V+) et la tension minimum est 0V.

Donc la plage de conversion est égale à la tension d'alimentation du PICAXE.

Néanmoins ces tensions de référence peuvent être modifiées en forçant les bits 14 et 15 dans l' instruction "adcsetup"

Si Bit14 = 1 alors Vref+ est la broche ADC3 / si Bit14 = 0 alors Vref+ = V+ (tension d'alimentation)

Si Bit15 = 1 alors Vref- est la broche ADC2 / si Bit15 = 0 alors Vref- = 0V

Exemple:

```
let adcsetup = 4    ; ADC0,1,2,3 en service
```

Section 3

PICAXE-20X2 (PIC18F14K22)

PICAXE-28X2-3V (PIC18F25K20)

PICAXE-40X2-3V (PIC18F45K20)

adcsetup est un masque

Avec ces modèles, le choix individuel des broches à utiliser comme entrée adc est possible.

Il suffit de placer le bit correspondant du masque en 1.

Bit 0 – ADC0	Bit 8 - ADC8
Bit 1 - ADC1	Bit 9 - ADC9
Bit 2 - ADC2	Bit 10 - ADC10
Bit 3 - ADC3	Bit 11 - ADC11
Bit 4 – ADC4	Bit 12 - ADC12
Bit 5 – ADC5	Bit 13 – *non utilisé*
Bit 6 – ADC6	Bit 14 - VRef+
Bit 7 - ADC7	Bit 15 - VRef- *(non disponible sur le modèle 20X2)*

Tensions de référence :

Par défaut, la tension maximum Vref+ est la tension d'alimentation (V+) et la tension minimum est 0V.

Donc la plage de conversion est égale à la tension d'alimentation du PICAXE.

Néanmoins ces tensions de référence peuvent être modifiées en forçant les bits 14 et 15 dans l' instruction "adcsetup"

Si Bit14 = 1 alors Vref+ est la broche ADC3 (28X2,40X2) ou ADC1 (20X2) / si Bit14 = 0 alors Vref+ = V+ (tension d'alimentation)

Si Bit15 = 1 alors Vref- est la broche ADC2 (28X2,40X2,non disponible sur 20X2) / si Bit15 = 0 alors Vref- = 0V

Exemple:

```
let adcsetup = %0000000000001111     ; ADC0,1,2,3 en service
```

Section 4

Concerne tous les modèles M2

adcsetup est un masque

Avec ces modèles, le choix individuel des broches à utiliser comme entrée adc est possible.

Il suffit de placer le bit correspondant du masque en 1.

Remarque : Sur les modèles M2, adcsetup est automatiquement mise à jour dés lors que l' on utilise une instruction "readadc", "readadc10 " ou "touch". Par conséquent cette instruction n' est utile que si l' on veut forcer une entrée en entrée digitale en forçant le bit correspondant en 0.

08M2

Bit 1 - ADC on C.1

Bit 2 - ADC on C.2

Bit 4 - ADC on C.4

14M2, 18M2, 20M2

Bit 0 - ADC on B.0	Bit 8 - ADC on C.0
Bit 1 - ADC on B.1	Bit 9 - ADC on C.1
Bit 2 - ADC on B.2	Bit 10 - ADC on C.2
Bit 3 - ADC on B.3	Bit 11 - ADC on C.3
Bit 4 - ADC on B.4	Bit 12 - ADC on C.4
Bit 5 - ADC on B.5	Bit 13 - ADC on C.5
Bit 6 - ADC on B.6	Bit 14 - ADC on C.6
Bit 7 - ADC on B.7	Bit 15 - ADC on C.7

Tensions de référence :

Par défaut, la tension maximum Vref+ est la tension d'alimentation (V+) et la tension minimum est 0V donc la plage de conversion est égale à la tension d'alimentation du PICAXE. Néanmoins ces tensions de référence peuvent être modifiées en utilisant l' instruction " adcconfig".

Exemple:

```
let adcsetup = %00001111          ; adc B.0-B.3 en service
```


(c) Revolution Education Ltd.
All rights reserved

2.3 backward

34

(tous les PICAXEs 18, 20, 28 et 40)

Syntaxe :

BACKWARD motor

- Motor est le moteur nommé A ou B.

Fonction :

Fait tourner un moteur à l'envers

Information :

Ceci est une pseudo commande élaborée pour des élèves plus jeunes possédant des modèles pré assemblés. En réalité elle est équivalente à « low4 :high5 » (moteur A) ou « low6 :high7 » (moteur B). Normalement cette commande n'est pas utilisée en dehors de la classe.

Exemple :

```
main :
            forward A      ; moteur A en marche avant
              wait 5         ; attendre 5 secondes
              backward A     ; moteur A en marche arrière
              wait 5         ; attendre 5 secondes
              halt A         ; arrêt du moteur A
              wait 5         ; attendre 5 secondes
              goto main      ; reboucler au début
```

2.4 bcdtoascii

35

(tous les PICAXEs)

Syntaxe:

BCDTOASCII variable, tens, units

BCDTOASCII wordvariable, thousands, hundreds, tens, units

La valeur à convertir est placée dans "variable" (0-99) ou dans "wordvariable" (0-9999)
"thousands" contient la valeur ascii du chiffre des dix-milliers
"hundreds" contient la valeur ascii du chiffre de centaines
"tens" contient la valeur ascii du chiffre des dizaines
"units" contient la valeur ascii du chiffre des unités

Commentaire :

Décompose un byte (valeur limitée à 99) ou un mot (valeur limitée à 9999) en une suite de valeurs ASCII

Les codes ASCII des chiffres vont de 48 pour zéro à 57 pour 9.

Exemple:

bcdtoascii b1,b2,b3

```
'par exemple si b1 = $23 alors b2 = 50 ( "2" ASCII ), b3 = 51 ( "3" ASCII )
```


(c) Revolution Education Ltd.
All rights reserved

2.5 bintoascii

36

(tous les PICAXEs)

Syntaxe:

BINTOASCII variable, hundreds, tens, units

BINTOASCII wordvariable, tenthousands, thousands, hundreds, tens, units

"variable" est la valeur d'un byte à convertir (0-255)
"wordvariable" est la valeur d'un mot à convertir (0-65535)
"tenthousands" contient la valeur ASCII du chiffre des dix-milliers
"thousands" contient la valeur ASCII du chiffre des milliers
"hundreds" contient la valeur ASCII du chiffre des centaines
"tens" contient la valeur ASCII du chiffre des dizaines
"units" contient la valeurASCII du chiffre des unités

Commentaire :

Décompose un byte ou un mot en une suite de valeurs ascii.

Les codes ascii des chiffres vont de 48 pour zéro à 57 pour 9.

Exemple:

bintoascii b1,b2,b3,b4

```
' par exemple : si b1 = 128 alors b2 = 49 ( "1" en ASCII ), b3 = 50 ( "2" en ASCII), b4 =
     56 ( "8" en ASCII )
```


(c) Revolution Education Ltd.
All rights reserved

2.6 booti2c

37

(uniquement les PICAXEs 20X2, 28X2 et 40X2)

Syntaxe:

Booti2c slot

- Slot est le numéro et l'adresse de l'emplacement de la mémoire EEPROM externe. (4 à 7)

Fonction:

Sur les PICAXE X2, il est possible de remplacer le programme interne par un nouveau programme se trouvant dans une EEPROM externe, connectée en I2C.

Information:

La commande **booti2c** peut être utilisée pour copier un programme depuis un emplacement d'une mémoire 24LC128 vers un emplacement de mémoire interne.

(NdT : Les 14M2, 18M2, 20M2 comportent chacun 2 emplacements mémoire (slots) de 2048 bytes. Les 28X2 et 40X2 comportent eux, 4 emplacements mémoire de 4096 bytes. Le 20X2 ne comporte lui qu'un seul slot de 4096 bytes- Ces emplacement mémoire sont indépendants et peuvent recevoir des programme différents. Les variables elles , sont partagées.)

La commande **booti2c** n'est traitée que si le numéro de révision du programme (fixé par la directive #revision lors du chargement du programme) de l'emplacement mémoire de l'EEPROM 24LC128 est supérieur à celui de l'emplacement mémoire interne actuellement.

Cela signifie que la copie de programme ne peut s'effectuer que si la nouvelle mémoire EEPROM 24LC128 est mise en place.

Si une mémoire n'est pas correctement mise, la donnée retournée par le circuit sera typiquement 0 ou 255 . Ces deux valeurs ne sont pas reconnues comme nombre #revision valides et seront ignorées.

Les paramètres de la commande booti2c ont pour format un unique byte qui représente l'adresse I2C de la mémoire externe, et son emplacement mémoire.

Bit7 24LC128 A2

Bit6 24LC128 A1

Bit5 24LC128 A0

Bit4 réservé pour un usage futur

Bit3 réservé pour un usage futur

Bit2 doit être à 1 pour utiliser l' i2c

Bit1, 0 numéro d'emplacement mémoire. (slot)

Les 2 bits de faible poids (0 et 1) sont recopiés à la même position au sein de la mémoire du programme interne.

La mémoire data reste inchangée.

Lors de la liaison I2C, la correspondance entre les valeurs d'emplacement mémoire devient : (en supposant une EEPROM d'adresse 0)

i2c slot		slot mémoire interne
4 (%00000100)	>	0 (%00000000)
5 (%00000101)	>	1 (%00000001)
6 (%00000110)	>	2 (%00000010)
7 (%00000111)	>	3 (%00000011)

Une fois que le programme a été copié, le microcontrôleur effectue un reset . Donc c'est le programme à l'emplacement 0 qui démarre.

Aussi, si vous souhaitez programmer une EEPROM avec un programme destiner à remplacer le programme de l'emplacement 2 sur une puce différente, la directive #slot 6 doit figurer lors du chargement de l'EEPROM.

L'EEPROM peut alors transférer le programme vers la cible.

Le type d'EEPROM doit être un modèle disposant d'une page tampon d'au moins 64 bytes. En conséquence, les EEPROM recommandées sont des Microchip 24LC128, ou 24LC256, ou 24LC512. Des mémoires d'autres marques peuvent ne pas fonctionner selon leurs spécifications de timing, ou de capacité de page tampon.

Exemple:

```
Booti2c 1       ; test l'EEPROM et charge l'emplacement 1 de la mémoire interne.
```

NdT : de fait, le test consiste à vérifier que le numéro de révision en EEPROM est supérieur à celui du programme interne dans l'emplacement 1, puis si c'est vérifié, le contenu de l'emplacement 5 de l'EEPROM sera recopié en emplacement 1 de la mémoire interne .

2.7 branch

39

(tous les PICAXEs)

Syntaxe:

BRANCH offset, (address0,address1...addressN)

- Offset est une variable ou une constante qui désigne une des adresses notées de 0 à N.
- Les adresses sont des labels qui spécifient la ligne vers laquelle il faut sauter

Commentaire :

Branche à l' adresse spécifiée (si elle existe).

Cette instruction permet de sauter à différentes portions du programme. Les adresses sont donc la destination des sauts.

Cette instruction a les mêmes effets que l' instruction ' On … goto ' .

Si 'offset' est plus grand que le nombre d'adresses spécifiées le branchement n'a pas lieu et le programme continue à l'instruction suivante.

Exemple:

```
reset1
let b1 = 0
     low B.0
     low B.1
     low B.2
     low B.3
main :
     inc b1
     if b1 > 4 then reset1
     branch b1,(btn0,btn1, btn2, btn3, btn4) ; suivant b1, sauter à btn0, btn1...

     btn0                ;label pour 'Offset = 0 '
          high B.0
          goto main
     btn1                ;label pour 'Offset = 1 '
          high B.1
          goto main
     btn2                ;label pour 'Offset = 2 '
          high B.2
          goto main
     btn3                ;label pour 'Offset = 3 '
          high B.3
          goto main
     btn4                ;label pour 'Offset = 4 '
          high B.4
          goto main
```

2.8 button

40 (tous les PICAXEs)

Syntaxe:

BUTTON pin, downstate, delay, rate, bytevariable, targetstate, address

' pin '	est une variable ou une constante qui spécifie la broche utilisée.
' dowstate '	est une variable ou une constante (0 ou 1) qui spécifie quel est l'état logique (0 ou 1) dans lequel se trouve le contact lorsque le bouton est pressé. Si l' entrée est en 1 (Valim) lorsque le bouton est pressé, 'downstate' = 1 sinon 'dowstate'= 0 .
'delay'	est une variable ou une constante (0, 1-254, 255) qui fixe le nombre de boucles a effectuer avant de lancer la fonction 'autorepeat' (si 'button' est utilisé dans une boucle). Si cette valeur est comprise entre 1 et 254, cette valeur sera chargée dans la variable 'bytevariable' lorsque le bouton est pressé puis décrémentée à chaque boucle. Lorsque le compteur atteint 0 on passe à la phase suivante (autorepeat). Ceci génère donc un retard . Si cette valeur est fixée à 255, la phase suivante (autorepeat) est ignorée. Seul le retard subsiste. Si cette valeur est fixée à 0, le retard est nul et la phase ' autorepeat ' est ignorée. Cette instruction est alors équivalente à ' ' if pin = targetstate then '
' rate '	est une variable ou une constante (0-255) qui spécifie le délai entre chaque boucle 'autorepeat'
'bytevariable'	est une variable utilisée pour compter le nombre de boucles 'autorepeat'. Elle doit être mise à 0 la première fois que l' on utilise l' instruction 'button' (avant la boucle qui contient 'button').
'targetstate'	(état cible) est une variable ou une constante (0 ou 1) qui spécifie dans quel état (0 = non pressé, 1 = pressé) le contact doit se trouver pour que le saut (goto) s' effectue à 'address'.
'address'	est un label qui spécifie où le programme doit sauter si le bouton est dans l' état cible

Commentaire :

Les contacts électromécaniques (commutateurs, interrupteurs, poussoirs, relais,...) sont sujets à des rebonds lorsqu' ils sont manœuvrés. La fermeture (l' ouverture) ne se fait pas franchement, il faut plusieurs millisecondes après l' action pour que leur état soit stable et définitif.

Suite à une action 'physique', le micro-contrôleur va donc enregistrer ces rebonds comme autant d'états successifs.

Une façon simple de résoudre ce problème, lorsque l' on a un seul circuit avec un ou deux contacts à traiter, est de mettre une pause. qui permet d'attendre que le contact soit stable et définitif. Par expérience, la valeur de ce retard sera facilement déterminée.

Mais lorsque l'on a des séries à réaliser, cette façon de faire ne convient pas. En effet chaque contact a des rebonds différents, même pour un même modèle, de la même série et du même constructeur.

L' utilisation de l'instruction 'button' s'impose alors.

Exemple:

```
init: b2 = 0        ; remise à zéro de l'octet cible avant de commencer la boucle
                    ; réalise une entrée sur C.0, active à l'état haut
                    ; et branche vers le label 'pushed' si C.0 passe à l'état 1

maBoucle: button C.0,1,200,100,b2,1,pushed  ; saut vers 'pushed' si C.0 = 1
     low B.7                                ; sortie à l'état bas
     pause 10                               ; petit delai
     goto maBoucle

pushed: high B.7 ; output on
     sertxd ("PUSH")                        ; message "touche enfoncée"
     goto myloop
```


(c) Revolution Education Ltd.
All rights reserved

2.9 calibadc (calibadc10)

42 (tous les PICAXEs M2 et X2, ainsi que le 20M et le 28X1)

Syntaxe:

CALIBADC variable

CALIBADC10 wordvariable

"variable" contient le résultat de la conversion analogique-digitale par un adc 8 bits de résolution, wordvariable par un adc 10 bits.

Commentaire :

Permet de calibrer un ADC en réalisant la conversion analogique-digitale d'une tension fixe interne connue.

0.6V pour les modèles 20M, 28X1, 40X1

1.2V pour les modèles 28X2-3V, 28X2-3V

1.024V pour tous les autres modèles qui supportent cette instruction.

Cette commande n' est pas disponible sur les modèles 28X2-5V/40X2-5V.

La tension de référence utilisée par les adc (readadc / readadc10) est la tension d'alimentation du PICAXE.

Dans le cas où cette tension d'alimentation est fournie par une batterie, la tension d'alimentation baisse au cours du temps : il en résulte que le résultat d'une conversion d'une même tension d'entrée diminue également au cours du temps.

L' instruction "calibadc/calibadc10" permet de corriger cette dérive en fournissant le résultat de la conversion analogique- digitale d'une tension de référence interne fixe. Ainsi, en utilisant périodiquement ces instructions vous pouvez mathématiquement corriger les baisses de la tension d'alimentation.

La relation, avec 0,6V comme tension de référence, est :

Valim = step * 6 / calib / 10

avec "step" = 255 pour calibadc / =1023 pour calibadc10

"calib" = valeur résultat de la conversion

*6/10 = 0,6 (tension de référence 0,6V).

Remarque : la tension d'alimentation nominale varie suivant les modèles de PICAXE.

Consultez la note d'application AN1702 sur le site de Microchip pour plus de détails sur comment utiliser cette instruction et comment la programmer.

Exemple:

```
calibadc b1              ; le résultat se trouve dans b1
```


(c) Revolution Education Ltd.
All rights reserved

2.10 calibfreq

43

(tous les PICAXEs sauf 08, 18, 18A, 18M, 28A & 28X)

Syntaxe :

CALIBFREQ {-} facteur

- facteur est une constante ou une variable comprise entre -15 et 15

Fonction :

Ajuster la fréquence de l'oscillateur interne du microcontrôleur. La valeur par défaut est 0.

Information :

Les PICAXE ont un oscillateur interne dont la fréquence est réglable par la commande « setfreq ».

Il est également possible d'ajuster cette fréquence. Les PICAXEs sont calibrés en usine et cette possibilité est inutile dans la plupart des applications. Cette commande permet par exemple d'affiner les fréquences pour améliorer les liaisons séries entre plusieurs PICAXEs. Elle affecte également toutes les commandes liées au temps et la variable système « time ».

Les valeurs positives augmentent la vitesse de l'oscillateur, les valeurs négatives la diminuent.

Cette commande doit être utilisée avec une extrême précaution, (limiter la variation entre -4 et +4 dans un premier temps). Elle peut placer la liaison série en dehors des tolérances et empêcher la reprogrammation du PICAXE, il faut dans ce cas procéder à un « hard reset » pour le reprogrammer.

En réalité, cette commande est une pseudo commande utilisant une commande « poke » pour modifier le registre OSCTUNE du microcontrôleur.

Pour les valeurs positives (de 0 à 15), le code équivalent basic est :

```
pokesfr OSCTUNE, facteur
pause 2
```

Pour les valeurs négatives (de -15 à -1), le code équivalent basic est :

```
let b12 = 64 - facteur
pokesfr OSCTUNE, b12
pause 2
```

Il faut noter que dans ce dernier cas, le byte b12 est utilisé et donc corrompu par cette commande. Ceci est nécessaire pour modifier le registre OSCTUNE.

2.11 clearbit

44

(PICAXEs 20X2, 28X1, 28X2, 40X1 & 40X2)

Syntaxe :

CLEARBIT var,bit

Var est la variable cible

Bit est le bit cible (0-7 pour les variables du type byte, 0-15 pour les mots)

Fonction :

Forcer à 0 un bit spécifique dans la variable.

Information :

Cette commande force à 0 un bit spécifique dans la variable cible.

Exemple :

```
clearbit b6, 0
Clearbit w4, 15
```

2.12 compsetup

45

(PICAXEs 20X2, 28X2 & 40X2)

Syntaxe:

COMPSETUP config , ivr

' config ' est une constante ou une variable fixant la configuration des comparateurs.

' ivr ' est une constante ou une variable spécifiant la tension de référence interne (échelle diviseur de tension résistif, voir schéma ci-dessous)

Commentaire :

Les modèles X2 comportent deux comparateurs, chacun pouvant comparer deux tensions analogiques fournies par les broches de sortie des ADC (Convertisseur Digital-Analogique) ou une tension fournie par une broche de sortie d'un ADC

et une tension de référence interne.

Plus précisément

- Pour un 20X2

le comparateur 1 compare ADC6 et ivr, le comparateur 2 compare ADC5 et (ivr OU ADC4)

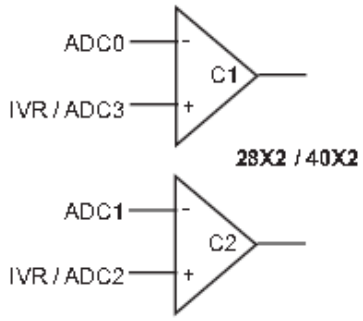
- Pour un 28X2 ou 40X2

le comparateur 1 compare ADC0 et (ivr OU ADC3), le comparateur 2 compare ADC1 et (ivr OU ADC2)

(Les ADC doivent être configurés par 'adcsetup ' avant d'utiliser cette instruction).

Plusieurs types de modèles X2 étant disponibles, ' Config ' est spécifique à chaque modèle, voir ci-dessous.

PICAXE-28X2-5V (PIC18F2520) et 40X2-5V (PIC18F4520)



Configuration :

bit7 *non utilisé, doit être mis en 0*

bit6 = 0 L' entréeVin+ du comparateur 1 est ADC3 / L' entrée Vin+ du comparateur 2 est ADC2
= 1 Les entrées Vin+ des deux comparateurs sont des tensions fournies par le diviseur de tension

bit5 *non utilisé, doit être mis en 0*

bit4 = 0 Un changement d'état de l' un ou l' autre des comparateurs ne modifie pas le drapeau ' compflag '
= 1 Un changement d'état de l' un ou l' autre des comparateurs place le drapeau ' compflag ' en 1

bit3 = 0 La sortie du comparateur 2 n' est pas inversée
= 1 La sortie du comparateur 2 est inversée

bit2 = 0 La sortie du comparateur 1 n' est pas inversée
= 1 La sortie du comparateur 1 est inversée

bit1 = 0 Le comparateur 2 est hors service
= 1 Les comparateurs 1 et 2 sont en service

bit0 = 0 Le comparateur 1 est hors service
= 1 Le comparateur 1 est en service

PICAXE-28X2 (PIC18F25K22) / 40X2 (PIC18F45K22)

PICAXE-28X2-3V (PIC18F25K20) / 40X2-3V (PIC18F45K20)

Configuration:

bit9 = 0 L' entrée Vin+ du comparateur 2 est la tension fournie par le diviseur de tension
= 1 L 'entrée Vin+ du comparateur 2 est la tension de référence (1,2 V)

bit8 = 0 L' entrée Vin+ du comparateur 1 est la tension fournie par le diviseur de tension
= 1 L 'entrée Vin+ du comparateur 1 est la tension de référence (1,2 V)

bit7 = 0 L' entrée Vin+ du comparateur 2 est ADC2
= 1 L' entrée Vin+ du comparateur 2 est soit la tension fournie par le diviseur de tension soit la tension de référence

bit6 = 0 L 'entrée Vin+ du comparateur 1est ADC3
= 1 L' entrée Vin+ du comparateur 1 est soit la tension fournie par le diviseur de tension soit la tension de référence

bit5 = 0 Un changement d'état du comparateur 2 ne modifie pas le drapeau ' compflag '
= 1 Un changement d'état du comparateur 2 place le drapeau ' compflag ' en 1

bit4 = 0 Un changement d'état du comparateur 1 ne modifie pas le drapeau ' compflag '
= 1 Un changement d'état du comparateur 1 place le drapeau ' compflag ' en 1

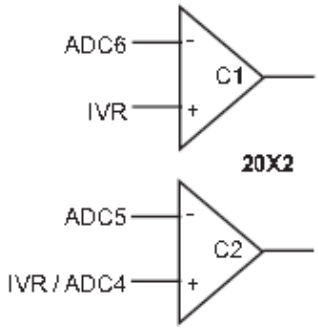
bit3 = 0 La sortie du comparateur 2 n' est pas inversée
= 1 La sortie du comparateur 2 est inversée

bit2 = 0 La sortie du comparateur 1 n' est pas inversée
= 1 La sortie du comparateur 1 est inversée

bit1 = 0 Le comparateur 2 est hors service
= 1 Le comparateur 2 est en service

bit0 = 0 Le comparateur 1 est hors service
= 1 Le comparateur 1 est en service

PICAXE-20X2



Configuration:

bit9 = 0 L' entrée Vin+ du comparateur 2 est la tension fournie par le diviseur de tension
= 1 L' entrée Vin+ du comparateur 2 est la tension de référence (1,024 V)

bit8 = 0 L' entrée Vin+ du comparateur 1 est la tension fournie par le diviseur de tension
= 1 L' entrée Vin+ du comparateur 1 est la tension de référence (1,024 V)

bit7 = 0 L' entrée Vin+ du comparateur 2 est ADC2
= 1 L' entrée Vin+ du comparateur 2 est soit la tension fournie par le diviseur de tension soit la tension de référence Co

bit6 = 1 non utilisé, doit être mis à 1

bit5 = 0 Un changement d'état du comparateur 2 ne modifie pas le drapeau ' compflag '
= 1 Un changement d'état du comparateur 2 place le drapeau ' compflag en 1

bit4 = 0 Un changement d'état du comparateur 1 ne modifie pas le drapeau ' compflag '
= 1 Un changement d'état du comparateur 1 place le drapeau ' compflag en 1

bit3 = 0 La sortie du comparateur 2 n' est pas inversée
= 1 La sortie du comparateur 2 est inversée

bit2 = 0 La sortie du comparateur 1 n' est pas inversée
= 1 La sortie du comparateur 1 est inversée

bit1 = 0 Le comparateur 2 est hors service
= 1 Le comparateur 2 est en service

bit0 = 0 Le comparateur 1 est hors service
= 1 Le comparateur 1 est en service

Etats des comparateurs :

Les états des deux comparateurs peuvent être lus à chaque instant dans la variable ' compvalue '

Les états des bits 0 et 1 sont les états des sorties des comparateurs :

– Le bit 0 est l' état de la sortie du comparateur 1.

Cette sortie peut être inversée, ce qui équivaut à croiser les entrées, en mettant en 1 le bit 2 de ' Config '

– Le bit 1 est l' état de la sortie du comparateur 2.

Cette sortie peut également être inversée, ce qui équivaut à croiser les entrées, en mettant en 1 le bit 3 de ' Config '

Au besoin, une modification de l' état des comparateurs peut entrainer une mise à 1 de ' compflag '

Lorsque cette modification est autorisée (bits 4 et 5 de ' Config ') la mise à 1 se fera à chaque basculement.

Ceci peut être utilisé pour déclencher une interruption (voir ' setinflags ').

Une modification peut aussi provoquer la sortie du mode ' sleep '.

Référence de tension interne (internal voltage reference)

Chaque entrée Vin- des comparateurs peuvent être comparées à une tension de référence interne. Cette tension, Vin+, peut être fournie (bits 6,7 de ' Config ' à 1) par un diviseur résistif interne.

Sur certains modèles , cette tension peut être fournie soit par un diviseur résistif interne soit par une tension de référence fixe (Bits 6,7,8,9 de ' Config ' à 1 pour un 28X2-PIC18F25K22 par exemple).

Le diviseur résistif interne est alimenté par la tension d'alimentation (voir schéma ci-dessous). Les résistances qui le composent agissent comme un simple diviseur de potentiel. Les résistances marquées ' 8R ' (résistances ' talon ') valent huit fois la valeur des autres résistances.

L 'octet ' ivr ' (internal voltage référence), attribut de l' instruction ' compsetup ', se configure comme suit suivant le modèle utilisé:

20X2, 28X2, 40X2

bit7 = 0 Le diviseur résistif est hors service
= 1 Le diviseur résistif est en service

bit6 non utilisé, doit être en 0

bit5 non utilisé, doit être en 0

bit4:0 Sélectionne quelle tension utiliser pour Vin+ parmi les 32 disponibles (N * Valim/32)
avec N = 0 à 31 (bits 4:0)

28X2-5V, 28X2-3V, 40X2-5V, 40X2-3V

bit7 = 0 Le diviseur résistif est hors service
= 1 Le diviseur résistif est en service

bit6 non utilisé, doit être en 0

bit5 = 0 La résistance ' talon' de valeur 8R est en service
= 1 La résistance ' talon ' de valeur 8R est court-circuitée (hors service)

bit4 non utilisé, doit être en 0

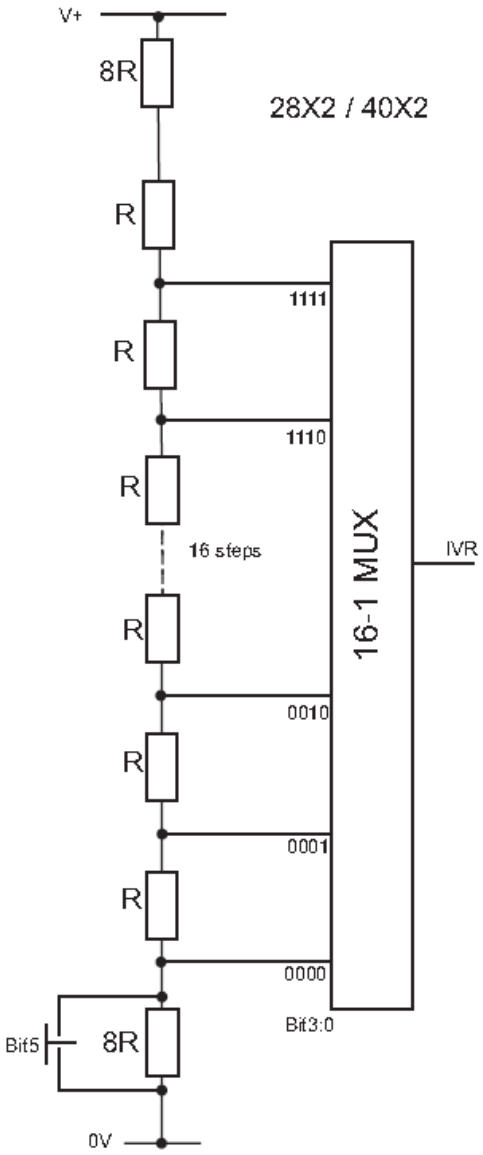
bit3:0 Sélectionne quelle tension utiliser pour Vin+ (*)

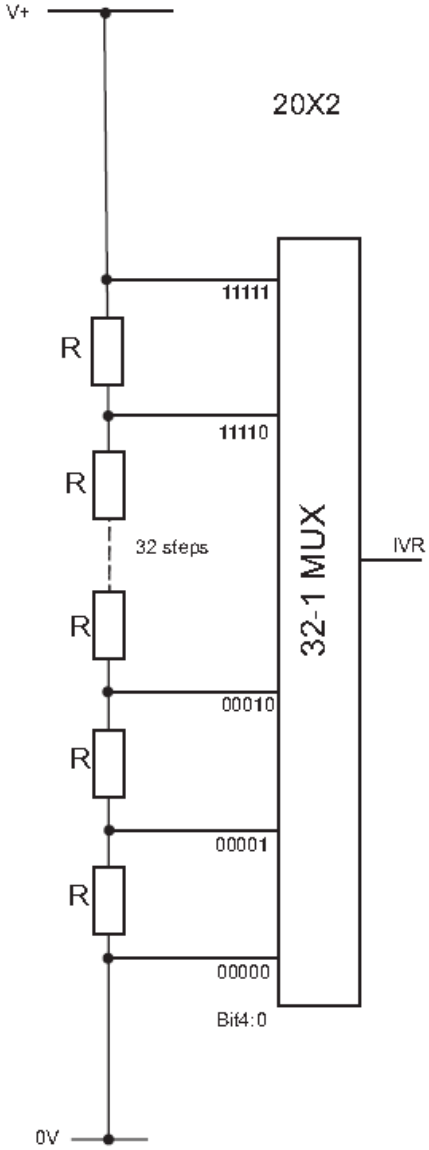
(*) N = 0 à 15 (bits 3:0)

Exemple:

```
init:
      adcsetup = 4              ; use adc 0-3 (28X2-5V)
      compsetup %00000011,0     ; comparateurs 1 et 2 en service
main:
      b1 = compvalue            ; lecture de la valeur
      debug                     ; affichage de la valeur
      pause 500                 ; courte pause
goto main                       ; retour
```


(c) Revolution Education Ltd.
All rights reserved





Si la résistance 'talon' est court-circuitée (bit 5 = 1), la tension de référence est

ivr = (N / 24) * Valim

Si la résistance 'talon' n'est pas court-circuitée (bit 5 = 0), la tension de référence est

ivr = (N / 32) * Valim + (Valim/4)

avec N Définit par les bits (bit3:bit0)

ivr = (N / 32) * Valim

avec N Définit par les bits (bit4:bit0)

(c) Revolution Education Ltd.
All rights reserved

2.13 count

50

(Tous les PICAXEs sauf 08, 18 & 28A)

Syntaxe :

COUNT pin, period, wordvariable

- Pin est une constante ou une variable spécifiant l'entrée utilisée.
- period est une constante ou une variable (1 à 65535ms à 4MHz).
- wordvariable est la variable de type word recevant le résultat du comptage (0 à 65535).

Fonction :

Compte les impulsions sur une broche entrée.

Information :

COUNT surveille l'entrée spécifiée et compte le nombre de fronts montants (de 0 vers 1) pendant le temps Définit par **"period"**. La variable recevant le résultat doit être de type word .

A la fréquence horloge de 4 MHz, l'entrée est vérifiée toutes les 20µs, la fréquence maximum des impulsions est alors de 25 kHz, en supposant que le signal d'entrée à un rapport cyclique de 50% (temps ON = temps OFF).

Attention aux contacteurs mécaniques, les rebonds et mauvais contacts provoquent des erreurs de comptage.

Count est une commande "bloquante", rien ne peut interrompre le temps de comptage, pas même une interruption.

Effets de la fréquence horloge :

La fréquence horloge détermine la période minimum du signal d'entrée.

Fréquence horloge	Période signal	Fréquence signal correspondante
4 MHz	40 µs	25 kHz
8 MHz	20 µs	50 kHz
16 MHz	10 µs	100 kHz
32 MHz	5 µs	200 kHz
64 MHz	2,5 µs	400 kHz

Le temps de comptage est aussi affecté par la fréquence horloge.

Fréquence horloge	Unité de temps de la variable « period»
4 MHz	1000 µs
8 MHz	500 µs
16 MHz	250 µs
32 MHz	125 µs
64 MHz	62,5 µs

Exemple :

```
debut :
```

```
Count C.1, 5000, w13      ; comptage des impulsions pendant 5 s
Debug                     ; affichage de la valeur
Goto debut                ; retour au début
```


(c) Revolution Education Ltd.
All rights reserved

data (eeprom)

61

Syntaxe:

DATA {localisation},(donnée,donnée…)

EEPROM {localisation},(donnée,donnée…)

- Localisation est une constante (optionnelle) entre 0 et 255 qui spécifie le début de l'enregistrement des données en EEPROM. Si la localisation n'est pas spécifiée, l'enregistrement commence à l'adresse 0, les enregistrements successifs se placent les uns à la suite des autres.
- Donnée est la constante (entre 0 et 255) enregistrée en mémoire EEPROM

Fonction:

Charge des données en mémoire EEPROM. Sans commande data ou eeprom, les mémoires prennent la valeur 0. Les commandes data et eeprom ont la même fonction et peuvent être utilisées indifféremment.

Information:

Il ne s'agit pas d'une instruction, mais d'une méthode de chargement de données en mémoire. Cette commande n'affecte pas la longueur du programme (sauf cas particuliers ci-dessous).

Particularité des PICAXEs 08M2 et 18M2 (pas le 18M2+):

Pour ces PICAXEs, la mémoire EEPROM est partagée avec les 2048 octets disponibles pour le programme. La première mémoire EEPROM est localisée en 2048, la seconde en 2047, etc. En conséquence, si les 256 octets de mémoire EEPROM sont utilisés, la longueur du programme doit être inférieure à 1792 octets.

Particularités des PICAXEs obsolètes:

PICAXE 28, 28A :	Localisation EEPROM de 0 à 63
PICAXEs 08,18,28X,40X :	Localisation EEPROM de 0 à 127

Exemple:

```
#PICAXE 14M2                    ;type de PICAXE utilisé
Data 0, ("Hello Word")    ;sauvegarde des 10 caractères en EEPROM
For b1 =0 to 10           ;début d'une boucle for/next
     Read b1,b2           ;lit le caractère localisé en b1 et l'enregistre dans b2
     Serout C.1,N2400,(b1)      ;transmission du caractère à l'afficheur série
Next b1                   ;boucle pour le caractère suivant, jusqu'à 10
end
```

2.14 daclevel

51

Syntaxe :

DACLEVEL niveau

- "niveau" est une variable/constante qui spécifie le niveau de sortie DAC (0-31)

Fonction :

Fixer le niveau de sortie DAC (32 niveaux, 0-31)

Information :

La commande daclevel est utilisée pour fixer le niveau de sortie DAC a l'un des 32 niveaux qui couvrent la plage de tension du DAC. Chaque niveau est donc 1/32 de la tension maximum. Une commande « readdac » peut aussi lire la valeur du DAC, ceci est équivalent à « readadc command on the DAC level ».

Une commande « dacsetup » doit avoir été utilisée avant que « daclevel » puisse fonctionner.

Exemple :

```
Init:       dacsetup %10100000        ; DAC externe ; tension d'alimentation

main:  for b1 = 0 to 31
         daclevel b1            ; règle le niveau du convertisseur DAC
         pause 1000

        next b1
        goto main               ; Retour au début
```


(c) Revolution Education Ltd.
All rights reserved

2.15 dacsetup

52

PICAXEs M2 et X2 suivants : 08M2,14M2, 18M2, 20M2, 28X2, 40X2

Syntaxe:

DACSETUP config

' config ' est une constante ou une variable qui spécifie la configuration du DAC (Convertisseur Analogique-Digital)

Commentaire :

Certains modèles de PICAXEs possèdent un convertisseur digital-analogique qui permet de fournir une tension de référence.

Cette tension peur être utilisée en interne ou en externe sur une broche (DAC output pin).

Dans ce dernier cas, l' utilisation d'un circuit " buffer " (un ampli OP monté en suiveur de tension par exemple) est impératif car cette sortie ne délivre que peu de courant (trop peu pour allumer une LED par exemple).

C' est strictement une sortie " tension de référence " .

Une fois le DAC configuré , une instruction ' daclevel ' doit être utilisée pour fixer la tension de référence (32 niveaux possibles). La tension maximum théorique est 31/32 ème de Valim soit 4,84 V avec Valim = 5 V.

Un ampli OP " MCP6022 " de Microchip donne de très bons résultats avec C = 100 nF.

Configuration :

bit7 = 0 DAC est hors service

= 1 DAC est en service

bit6 : non utilisé, doit être mis en 0

bit5 = 0 DAC en interne seulement

= 1 DAC en interne et en externe.

bit4 = 0 non utilisé,doit être en 0

bit3-2 = 00 La tension de référence Vref+ est la tension d'alimentation (Valim)

= 01 La tension de référence Vref+ est la broche Vref+ (voir adcconfig)

= 10 La tension de référence Vref+ est la tension fvr (voir fvrsetup)

= 11 non utilisé

bit1 = 0 non utilisé, doit être en 0

bit0 = 0 la tension de référence Vref- est la masse (0V)

= 1 La tension de référence Vref- est la broche Vref-

Exemple:

```
init:
      low DAC_PIN          ; force la broche DAC en sortie
      dacsetup %10100000   ;DAC en service,sortie externe,Vref+ = Valim
main:
      for b1 = 0 to 31
            daclevel b1    ; V = b1 * Valim * (31 / 32)
            pause 1000
      next b1
```


(c) Revolution Education Ltd.
All rights reserved

```
goto main           ; retour
```


(c) Revolution Education Ltd.
All rights reserved

2.16 debug

54

Syntaxe:

DEBUG {var}

- var est une variable optionnelle (par exemple b0), cette variable est sans importance et n'existe uniquement que pour la compatibilité avec les anciens programmes.

Fonction:

Affiche des informations de variables dans une fenêtre de débogage.

Information:

La commande Debug envoie les valeurs des variables utilisées par le PICAXE par l'intermédiaire du câble de téléchargement. L'affichage des variables s'effectue sur l'écran de votre ordinateur dès le téléchargement du programme. Cette commande ralentit considérablement le programme et il est conseillé de limiter cette commande pour le débogage.

Pour afficher des messages définitts par l'utilisateur, il est conseillé d'utiliser la commande « sertxd ».

A noter que les PICAXEs de la série 08 et 14 couplant la sortie TXD avec la sortie B0 sont susceptibles de corrompre la valeur des données série. Dans ce cas là, il est recommandé d'utiliser la structure suivante avant une commande debug.

```
low B.0         ; positionne B.0 à l'état bas pour éviter la corruption de donnée
pause 500       ; attente de 500ms
debug           ; affichage des valeurs sur l'écran de l'ordinateur
```

Exemple:

```
debut:
        inc b1              ; incrémentation de b1
        readadc A.2,b2      ; lecture valeur analogique sur A.2 positionne valeur dans b2
        debug               ; affichage des valeurs sur l'écran de l'ordinateur
        pause 500           ; attente de 500ms
        goto debut          ; aller à debut, bouclage du programme
```

2.17 dec

55

Syntaxe:

DEC variable
- 'variable' est la variable à décrémenter.

Fonction:
Décrémente (soustrait une unité) à la variable indiquée.

Information:

Cette commande est une simplification pour : **let variable = variable - 1**

Exemple:

```
let b2 = 10
for b1 = 1 to 5
     dec b2              ; b2 prends successivement les valeurs 9, 8, 7, 6 et 5
next b1
```


(c) Revolution Education Ltd.
All rights reserved

2.18 disablebod

56

Syntaxe:

DISABLEBOD

Fonction:

Désactive la détection matérielle des chutes de tension.

Information:

Certains PICAXEs disposent d'une fonction interne de détection matérielle de chute de tension, ceci afin de redémarrer automatiquement et proprement le PICAXE en cas de chute de tension d'alimentation. (une baisse de tension en dessous d'un certain seuil, même pendant un temps bref peut provoquer un fonctionnement erratique de n'importe quel microprocesseur).

La détection de chute de tension est toujours activée par défaut quand un programme est lancé. Néanmoins, il est parfois utile de désactiver cette fonction afin de réduire le courant consommé par une application alimentée sur piles lorsque la puce est "endormie".

La tension minimale est fixée pour chaque PICAXE de la façon suivante :

1.8V 28X2-3V, 40X2-3V
1.9V 20X2, 14M2, 18M2, 20M2, 28X2, 40X2
2.1V 08, 08M, 14M, 20M, 28X1, 40X1
2.3V 08M2
3.2V 28X2-5V, 40X2-5V
None 18, 18A, 18M, 18X, 28A, 28X, 40X

L'utilisation de la commande **disablod** avant la commande **sleep** réduira considérablement la consommation d'énergie pendant la période où la puce est endormie.

Exemple:

```
main:
     disablebod          ; désactive la détection de chute de tension
     sleep 10            ; sommeil pendant 23 secondes (2.3x10)
     enablebod           ; active la détection de chute de tension
     goto main           ; boucle au début
```


(c) Revolution Education Ltd.
All rights reserved

2.19 disabletime

57

Syntaxe:

DISABLETIME

Fonction:
Désactive le compteur de temps

Information:

Les PICAXEs de la série M2 disposent d'un compteur de temps, à savoir : une variable de type word qui est automatiquement incrémentée une fois par seconde. Ce compteur de secondes est automatiquement démarré lors du démarrage ou du re-démarrage du PICAXE, mais peut également être activé ou désactivé au moyen des commandes **enabletime** et **disabletime**.

Effet de l'augmentation de vitesse de l'horloge du PICAXE

La fonction time fonctionnera correctement à 4MHz et à 16 Mhz.

A 2MHz ou à 8MHz, l'intervalle sera de 2s

A 32 MHz, l'intervalle sera de 0,5s

Exemple:

```
main:
     pause 5000
     disabletime          ; désactive le compteur de temps
     pause 5000           ; attends 5 secondes
     enabletime           ; active le compteur de temps
     debug                ; Affiche la valeur du compteur de temps

     goto main            ; boucle au début
```


(c) Revolution Education Ltd.
All rights reserved

2.20 disconnect

58

Syntaxe:

DISCONNECT

Fonction:

Déconnecte le PICAXE de sorte qu'il ne vérifie plus l'éventualité du téléchargement d'un nouveau programme.

Information:

Les PICAXEs observent très fréquemment le port série de téléchargement au cas où un ordinateur tenterait de démarrer le téléchargement d'un nouveau programme. Néanmoins, si vous désirez utiliser ce port pour faire des communications série (au moyen de la commande serrxd), il est nécessaire de désactiver cette vérification périodique. Notez toutefois que l'utilisation de la commande **serrxd** provoque l'envoi préalable et automatique d'une commande **disconnect**.

Une fois que la commande disconnect a été utilisée, il n'est plus possible de télécharger un nouveau programme jusqu'à ce que :

1) la commande **reconnect** soit lancée,

2) la commande **reset** soit lancée

3) Un redémarrage matériel soit réalisé.

Si vous avez fait en sorte qu'il ne semble plus possible de télécharger un nouveau programme, souvenez-vous qu'il est toujours possible de le faire au moyen de la procédure de redémarrage matériel.

Exemple:

```
serrxd [1000, timeout],@ptrinc,@ptrinc,@ptr
reconnect
```

2.21 do ... loop

59

Syntaxe:

```
DO
{code}
LOOP UNTIL/WHILE VAR ?? COND

DO
{code}
LOOP UNTIL/WHILE VAR ?? COND AND/OR VAR ?? COND...

DO UNTIL/WHILE VAR ?? COND
{code}
LOOP

DO UNTIL/WHILE VAR ?? COND AND/OR VAR ?? COND...
{code}
LOOP
```

- VAR est la variable a tester
- COND est la condition du test
- Les points d'interrogation représentent n'importe quelles conditions suivantes:
- = Egal à
- is Egal à
- <> Différent de
- != Différent de
- > Plus grand que
- < Plus petit que

Fonction:

Loop vérifie la validité d'une condition. Tant que la condition est vraie, avec While, ou jusqu'à ce qu'elle soit fausse avec Until

Information :

Ce type d'instruction crée une boucle, qui sera répétée, tant que ou jusqu'à ce que la condition spécifiée dans DO soit vérifiée.

La commande Exit peut être utilisée, pour sortir à tout moment de la boucle DO ... LOOP.

Exemple:

```
Do                                  'Départ
     high B.1                       'Port B.1 a l'état haut
     pause 1000                     'pause une seconde
     low B.1                        'port B.1 a l'état bas
     pause 1000                     'pause une seconde
     inc b2                         'b2=b2+1
     if pinC.1 = 1 then exit        'Test du port C.1, sortie s'il est à 1
loop while b2 < 5                   'On continue, tant que b2 ne vaut pas 5
```


(c) Revolution Education Ltd.
All rights reserved

2.22 doze

60

Syntaxe:

DOZE periode

- periode est une variable ou une constante qui précise la durée de l'endormissement partiel (avec périphériques actifs).

Fonction:

Patiente un moment.

La consommation d'énergie est réduite, mais la précision de chronométrage est altérée. La commande **doze** utilise la même fin de chronométrage (timeout) que la commande **sleep** (à savoir 2,1s)

Information:

La commande **doze** place le PICAXE dans un mode de consommation réduite pendant une durée courte (comme la commande **sleep**) Toutefois, à l'inverse de la commande sleep, les chronomètres (timers) restent actifs et donc les commandes **pwmout**, **timer** et **servo** continueront de fonctionner. La période de temps nominale est de 2,1s mais en raison des tolérances de fabrication des PICAXEs, cette durée peut varier de -50% à +100%. La température externe affecte également cette précision, et en conséquence aucun projet qui nécessite une bonne précision de la mesure du temps ne doit utiliser la commande **doze**.

doze 0 place le microcontrôleur en mode de consommation réduite permanent. Il ne se réveillera plus toutes les 2,1 s

Les deux seules façons de le réveiller sont :

- l'utilisation d'une interruption matérielle (à savoir capturer un changement de tension sur une broche, ou une fin de chronométrage matériel (=timer tick))
- l'utilisation de la procédure de redémarrage matériel

Le téléchargement de nouveaux programmes ne fonctionne plus lorsque le microcontrôleur est en mode de consommation réduite permanent.

Effet de l'augmentation de vitesse de l'horloge du PICAXE

La commande **doze** utilise un chronomètre interne différent de l'horloge du PICAXE (résonateur). Elle n'est donc pas affectée par un changement de cette dernière.

Exemple:

```
main:
     high B.1            ; Met sous tension la broche B.1
     doze 1              ; patiente pendant 2.1 s
     low B.1             ; Met hors tension la broche B.1
     doze 1              ; patiente pendant 2.1 s

     goto main           ; boucle au début
```

2.23 eeprom (data)

61

Syntaxe:

DATA {localisation},(donnée,donnée…)

EEPROM {localisation},(donnée,donnée…)

- Localisation est une constante (optionnelle) entre 0 et 255 qui spécifie le début de l'enregistrement des données en EEPROM. Si la localisation n'est pas spécifiée, l'enregistrement commence à l'adresse 0, les enregistrements successifs se placent les uns à la suite des autres.
- Donnée est la constante (entre 0 et 255) enregistrée en mémoire EEPROM

Fonction:

Charge des données en mémoire EEPROM. Sans commande data ou eeprom, les mémoires prennent la valeur 0. Les commandes data et eeprom ont la même fonction et peuvent être utilisées indifféremment.

Information:

Il ne s'agit pas d'une instruction, mais d'une méthode de chargement de données en mémoire. Cette commande n'affecte pas la longueur du programme (sauf cas particuliers ci-dessous).

Particularité des PICAXEs 08M2 et 18M2 (pas le 18M2+):

Pour ces PICAXEs, la mémoire EEPROM est partagée avec les 2048 octets disponibles pour le programme. La première mémoire EEPROM est localisée en 2048, la seconde en 2047, etc. En conséquence, si les 256 octets de mémoire EEPROM sont utilisés, la longueur du programme doit être inférieure à 1792 octets.

Particularités des PICAXEs obsolètes:

PICAXE 28, 28A : Localisation EEPROM de 0 à 63

PICAXEs 08,18,28X,40X : Localisation EEPROM de 0 à 127

Exemple:

```
#PICAXE 14M2                ;type de PICAXE utilisé
Data 0, ("Hello Word")      ;sauvegarde des 10 caractères en EEPROM
For b1 =0 to 10             ;début d'une boucle for/next
      Read b1,b2            ;lit le caractère localisé en b1 et l'enregistre dans b2
      Serout C.1,N2400,(b1) ;transmission du caractère à l'afficheur série
Next b1                     ;boucle pour le caractère suivant, jusqu'à 10
end
```


(c) Revolution Education Ltd.
All rights reserved

2.24 enablebod

62

Syntaxe:

ENABLEBOD

Fonction:

Active la détection matérielle des chutes de tension.

Information:

Certains PICAXEs disposent d'une fonction interne de détection matérielle de chute de tension, ceci afin de redémarrer automatiquement et proprement le PICAXE en cas de chute de tension d'alimentation. (une baisse de tension en dessous d'un certain seuil, même pendant un temps bref peut provoquer un fonctionnement erratique de n'importe quel microprocesseur).

La détection de chute de tension est toujours activée par défaut quand un programme est lancé. Néanmoins, il est parfois utile de désactiver cette fonction afin de réduire le courant consommé par une application alimentée sur piles lorsque la puce est "endormie".

La tension minimale est fixée pour chaque PICAXE de la façon suivante :

1.8V 28X2-3V, 40X2-3V
1.9V 20X2, 14M2, 18M2, 20M2, 28X2, 40X2
2.1V 08, 08M, 14M, 20M, 28X1, 40X1
2.3V 08M2
3.2V 28X2-5V, 40X2-5V
None 18, 18A, 18M, 18X, 28A, 28X, 40X

L'utilisation de la commande **disablod** avant la commande **sleep** réduira considérablement la consommation d'énergie pendant la période où la puce est endormie.

Exemple:

```
main:
      disablebod          ; désactive la détection de chute de tension
      sleep 10            ; sommeil pendant 23 secondes (2.3x10)
      enablebod           ; active la détection de chute de tension
      goto main           ; boucle au début
```


(c) Revolution Education Ltd.
All rights reserved

2.25 enableTime

63

Syntaxe:

ENABLETIME

Fonction:
Active le compteur de temps

Information:

Les PICAXEs de la série M2 disposent d'un compteur de temps, à savoir : une variable de type word qui est automatiquement incrémentée une fois par seconde. Ce compteur de secondes est automatiquement démarré lors du démarrage ou du re-démarrage du PICAXE, mais peut également être activé ou désactivé au moyen des commandes **enabletime** et **disabletime**.

Effet de l'augmentation de vitesse de l'horloge du PICAXE

La fonction time fonctionnera correctement à 4 MHz et à 16 Mhz.

A 2 MHz ou à 8 MHz, l'intervalle sera de 2s

A 32 MHz, l'intervalle sera de 0,5s

Exemple:

```
main:
     pause 5000
     disabletime          ; désactive le compteur de temps
     pause 5000           ; attends 5 secondes
     enabletime           ; active le compteur de temps
     debug                ; Affiche la valeur du compteur de temps

     goto main            ; boucle au début
```


(c) Revolution Education Ltd.
All rights reserved

2.26 end

64

Syntaxe:

END

Fonction:

Passe définitivement en mode sommeil jusqu'a nouvel arrêt-redémarrage, ou nouvelle reprogrammation via cordon PC.

La consommation est réduite à son minimum (en supposant qu'aucune sortie ne soit connectée à une charge), les timers sont arrêtés.

Information :

La commande end force le µP en mode basse consommation à la fin d'un programme.

A noter : Comme le compilateur place automatiquement un end à la fin du programme, cette commande est rarement nécessaire

La commande end, bloquant le (les) timer(s), certaines commandes nécessitant ces mêmes timers, telles que SERVO, ou PWMOUT seront donc bloquées, après le end.

Si vous ne souhaitez pas ce type de blocage, il faudra placer une commande STOP en fin de programme. Celle-ci ne forcera pas le µP en mode basse consommation.

La principale utilité de la commande end est de séparer le programme principal des sous routines, comme dans l'exemple ci-dessous. Ceci évite de voir le programme risquer de se bloquer, en tournant inDéfinitment dans une sous routine.

Exemple:

```
main:
let b2 = 15        ; défini la valeur b2
pause 2000         ; attente 2 secondes
gosub flsh         ; appel sous-procédure flsh
let b2 = 5         ; redéfinit la valeur b2
pause 2000         ; attente 2 secondes
end                ; arrêt en cas de plantage de la sous procédure

flsh:
for b0 = 1 to b2   ; boucle deux passages
high B.1           ; passe B.1 à l'état haut
pause 500          ; attente 0.5 secondes
low B.1            ; passe B.1 à l'état bas
pause 500          ; attente  0.5 secondes
next b0            ; fin de la boucle
return             ; retour au programme principal
```

2.27 exit

65

Syntaxe:

EXIT

Fonction:

Exit est utilisée afin de clore immédiatement une boucle Do...Loop ou For...Next.

Information :

La commande exit “sort” le pointeur de la boucle, tout comme le ferait la commande Goto N° ligne, ou Goto Label.

Exemple:

```
main:
      do                          ; démarre boucle
      if b1 = 1 then              ; teste valeur b1
            exit                  ; si b1=1 sortie de boucle
      end if                      ; fin du test
      loop                        ; poursuit la boucle
```


(c) Revolution Education Ltd.
All rights reserved

2.28 for...next

66

Syntaxe:

FOR variable = debut TO fin {STEP {-} au pas de}

(autres lignes de programme)

NEXT {variable}

- Variable est utilisée comme compteur de boucle
- Début est la valeur de départ de Variable
- Fin est la valeur de fin de variable
- "Au pas de" est l'incrément (optionnel) du comptage. Le pas par défaut est de 1. Si l'incrément est spécifié négatif, Début sera supérieur à Fin, la boucle fonctionnera donc en mode décomptage.

Fonction:

Toute section de code placée entre For et Next sera exécutée autant de fois que la valeur Fin.

Information :

La commande For ...Next est utilisée pour répéter l'exécution des lignes de codes placées entre For et Next. Si, pour le comptage maximum (fin) une variable de type byte est utilisée, les instructions, entre for et next pourront être répétées au plus 255 fois. A chaque passage, la valeur de comptage est incrémentée (ou décrémentée) de la valeur 1, par défaut.

Lorsque celle-ci dépasse la valeur fixée par fin , la boucle de comptage s'arrête, et le programme reprend, après l'instruction next.

Important: l'imbrication maximum de boucles For...Next est de 8. Pensez a bien utiliser des variables différentes, pour chaque Définition de boucle.

Il est possible de sortir d'une boucle For...Next, à tout moment, par l'utilisation de la commande Exit

Exemple:

```
main:
     for b0 = 1 to 20                ; défini boucle 20 fois
           if pinC.1 = 1 then exit   ; test état port C.1 et sort, si égal 1
           high B.1                  ; passe B.1 à l'état haut
           pause 500                 ; attente 0.5 secondes
           low B.1                   ; Passe B.1 à l'état bas
           pause 500                 ; attente 0.5 secondes
     next b0                         ; fin de boucle
     pause 2000                      ; attente for 2 secondes
     goto main                       ; retour en tête
```


(c) Revolution Education Ltd.
All rights reserved

2.29 forward

67

Syntaxe :

FORWARD moteur

- Moteur est le moteur commandé, nommé soit A, soit B

Fonction:

Initialise une sortie commande moteur, mode marche avant.

Information :

Il s'agit là d'une 'pseudo' commande, destinée aux plus jeunes utilisateurs. Elle est équivalente à 'high 4 : low 5 (pour le moteur A) ou 'high 6 : low 7' pour le moteur B. Cette commande n'a pas lieu d'être hors salles de classes.

Exemple:

```
main:
      forward A           ; moteur marche avant
      wait 5              ; attente 5 secondes
      backward A          ; moteur marche arrière
      wait 5              ; attente 5 secondes
      halt A              ; inversion moteur A
      wait 5              ; attente 5 secondes
      goto main           ; retour début
```


(c) Revolution Education Ltd.
All rights reserved

2.30 fvrsetup

68

(Picaxes 08M2, 14M2, 18M2, 20M2, 28X2, 40X2)

Syntaxe:

FVRSETUP OFF

FVRSETUP config

"config" est une constante ou une variable spécifiant une tension de référence fixe.

Commentaire :

Les modèles ci-dessus comportent des circuits de référence de tension fixe.

Ils peuvent être mis hors service ou réglés à l' une des trois valeurs suivantes :

1,024 V (fvr1024)

2,048 V (fvr2048)

4,096 V (fvr4096)

Remarque 1 :

la tension de référence fixe (FVR) ne peut pas être supérieure à la tension d'alimentation (ainsi 4,096 V n' est disponible que si la tension d'alimentation est 5 V).

Remarque 2 :

la tension de référence 1,024V (FVR1024) ne peut pas être utilisée comme tension de référence maximum (Vref+) dans un ADC, seules les tensions 2,048V (FVR2048) et 4,096V (FVR4096) sont utilisables (voir adcconfig)

Remarque 3 :

pour limiter la consommation, les circuits FVR sont automatiquement mis hors service aprés une instruction "readadc". S' il en est besoin à nouveau, il faut donc les remettre en service par une nouvelle commande "fvrsetup"

Remarque 4 :

une instruction "calibadc" place automatiquement la tension de référence à 1,024 V (fvr1024).

Remarque 5 :

cette commande peut être également utilisée pour fixer la tension de référence d'un DAC (Digital Analogique Converter).

voir la commande "dacsetup"

Exemple:

```
fvrsetup FVR1024          ; la tension de référence est 1,024V
```


(c) Revolution Education Ltd.
All rights reserved

fvrsetup

68

Syntaxe :

FWRSETUP OFF

FWRSETUP configuration

- Configuration est une variable, ou une constante, qui spécifie la valeur de la tension de référence de la puce.

Fonction:

Configure la valeur de référence de tension interne.

Information :

Certaines puces disposent d'une référence interne de tension.

Celle-ci peut-être, soit désactivée, soit fixée à l'une des valeurs suivantes, selon la constante utilisée :

FVR1024	1.024V
FVR2048	2.048V
FVR4096	4.096V

- A noter: la référence ne peut excéder la tension d'alimentation, donc, la valeur 4.096 n'est valide, QUE pour une tension d'alimentation de 5V).
- A noter également, que la référence à 1.024 ne peut être utilisée comme référence ADC (seules 2.048 et 4.096 peuvent l'être). Voir la commande ADCCONFIG pour plus de détails. Dans le but de réduire la consommation du système, l'utilisation de la la commande FVR est automatiquement annulée après une lecture ADC, il est donc nécessaire de réimplémenter la commande, après l'utilisation d'un readadc, si cette fonctionnalité est de nouveau nécessaire.
- De plus, la tension de référence est remise à 1.024V, après chaque commande calibadc.
- La commande FWR peut également être utilisée comme référence, pour la conversion digitale-analogique (voir la commande DACsetup, pour plus de détails).

Exemple:

```
fvrsetup FVR1024          ; fixe la référence à 1.024V
```


(c) Revolution Education Ltd.
All rights reserved

2.31 get

69

Syntaxe :

GET adresse,variable, variable, variable 16 bits ...

- Adresse est une variable, ou une constante, qui spécifie une adresse dans le scratchpad. Les valeurs valides sont :

  0 à 127 Pour les puces type X1

  0 à 127 Pour les puces 20X2

  0 à 1023 Pour les autres puces X2

Fonction:

Lire une donnée inscrite en scratchpad..

Information :

La fonction de la commande get est de lire une donnée stockée temporairement dans le scratchpad. Cette fonctionnalité évite la “consommation” d’emplacements mémoire habituellement utilisés par b0, b1, etc ..

Les commandes put et get n’ayant aucun effet sur le pointeur du scratchpad, les commandes suivantes utilisées par le pointeur indirect (ptr) ne seront pas modifiées, par ces commandes

Lorsqu'une variable de type 16 bits (WORD) est utilisée, les deux octets de cette variable sont enregistrés/récupérés de la manière suivante : octet de poids le plus bas, puis octet de poids le plus haut, (adresse suivante à adresse+1).

Exemple:

```
get 1,b1                ; copie la valeur du registre 1 dans b1
get 1, word w1
```

2.32 gosub (call)

70

Syntaxe:

GOSUB adresse

-adresse est la destination du saut

Fonction:

Déplace le pointeur vers une sous-routine située à adresse, puis, revient automatiquement une ligne de code plus loin.

Le compilateur accepte indifféremment les commandes Gosub ou Call.

Information :

La commande Gosub (“aller à une sous-procédure”) est un saut temporaire vers une sous-routine . Une fois cette dernière effectuée, le programme se poursuivra juste après l’instruction Gosub, SOUS RESERVE que la sous-routine se termine par un Return. Ne pas confondre Gosub, qui est un saut temporaire, avec retour obligatoire, avec Goto, qui est un saut définitif.

Le tableau ci-dessous permet de connaitre le nombre maximum d’imbrications d’appels Gosub, suivant les µP utilisés.

Par défaut, tout PICAXE accepte au moins 8 imbrications.

Nombre maximum d’imbrications, selon hardware :

	Gosubs	Interruptions	Pile
Toutes les puces ‘M2’	255	1	8
Toutes les puces ‘X2’	255	1	8
Toutes les puces ‘X1’	255	1	8
Toutes les puces ‘X’ (obsolètes)	255	1	4
Toutes les puces ‘M’	15	1	4
Toutes les puces ‘A’ (obsolètes)	16	0	4

En cas de multitâche, pour les puces M2, chaque tâche a son propre maximum d’imbrication fixé à 8.

Les sous-routines sont habituellement utilisées pour réduire la taille du code, en implémentant des instructions répétitives au sein d’une même procédure, en y passant les variables adaptées. Voir l’exemple ci-dessous.

```
main:
  let b2 = 15            ; définit valeur b2
  gosub flsh             ; appel sous procédure slh
  end                    ; stop le programme en cas d'erreur en sous-procédure

flsh:
  for b0 = 1 to b2       ; Définit boucle, deux fois
  high B.1               ; met le port B.1 à l’état haut
  pause 500              ; attente 0.5 secondes
  low B.1                ; met le port B.1 à l’état bas
  pause 500              ; attente 0.5 secondes
  next b0                ; fin de la boucle
  return                 ; sortie de la sous-procédure
```

2.33 goto

71

Syntaxe:

GOTO adresse

- adresse est la destination du saut

Fonction:

Déplace le pointeur vers adresse (label ou No ligne)

Information :

La commande goto est un saut définitif vers une autre zone de programme (label ou No de ligne).

Exemple :

```
main:
   high B.1              ; met le port B.1 à l'état haut
   pause 5000            ; attente 5 secondes
   low B.1               ; met le port B.1 à l'état bas
   pause 5000            ; attente 5 secondes
   goto main             ; retour en tête
```


(c) Revolution Education Ltd.
All rights reserved

2.34 hi2cin

72

Syntaxe:

HI2CIN (variable,…)

HI2CIN location, (variable,…)

HI2CIN [newslave], (variable,…) **(seulement PICAXE X2)**

HI2CIN [newslave], location, (variable,…) **(seulement PICAXE X2)**

- variable(s) dans laquelle la lecture de l'octet du périphérique i2c se positionne.
- location est une variable/constante spécifiant l'emplacement de départ de la lecture.
- newslave est une option pour une nouvelle adresse de commande.

Fonction:

Permet de lire les données du périphérique i2c et de la positionner dans une variable (ou plusieurs variables).

Information:

Les données sont lues à partir de l'adresse de départ indiquée, la lecture des autres données séquentielles est possible si le périphérique i2c le supporte.

L'emplacement doit être un octet ou un mot Définit par la commande i2csetup La commande i2csetup doit donc être Définit avant cette commande. Si l'on s'adresse à plusieurs périphériques i2c, il peut être nécessaire de changer l'adresse de l'esclave i2c en utilisant les options facultatives [newslave].

Si les périphériques i2c sont mal configurés ou que des données i2cslave erronées ont été utilisées, la valeur 255 ($FF) est chargée dans chaque variable.

Exemple:

```
; Utilisation d'un module horloge DS1307
; ce module utilise le format BCD (voir utilisation de BCDTOASCII et BINTOASCII)
; le PICAXE est maître, le module DS1307 est l'esclave et utilise l'adresse i2c %11010000

      hi2csetup i2cmaster, %11010000, i2cslow, i2cbyte ; adressage DS1307, mode
                                                       ;slow en octet
debut:
      hi2cin 0,(b0,b1,b2,b3,b4,b5,b6,b7)               ; lecture et affichage de
                                       ;l'heure et de la date
      debug                      ; affichage des valeurs sur l'écran de l'ordinateur
      pause 2000                 ; attente de 2 secondes
      goto debut                 ; aller à debut, bouclage du programme
```

2.35 hi2cout

74

Syntaxe:

HI2COUT location, (variable,…)

HI2COUT (variable,…)

HI2COUT [newslave], location, (variable,…) **(seulement PICAXE X2)**

HI2COUT [newslave], (variable,…) **(seulement PICAXE X2)**

- variable contient l'octet à écrire.
- location est une variable/constante spécifiant l'emplacement de départ de l'écriture.
- newslave est une option pour une nouvelle adresse de commande.

Fonction:

Permet d'envoyer des données sur le bus i2c.

Information:

L'envoi d'une donnée s'effectue à partir de l'adresse de départ indiquée, l'envoi d'autres données séquentielles est possible si le périphérique i2c le supporte.

L'emplacement doit être un octet ou un mot Définit par la commande i2csetup La commande i2csetup doit donc être Définit avant cette commande. Si l'on s'adresse à plusieurs périphériques i2c, il peut être nécessaire de changer l'adresse de l'esclave i2c en utilisant les options facultatives [newslave].

Exemple:

```
; Utilisation d'un module horloge DS1307
; le PICAXE est maître, le module DS1307 est l'esclave et utilise l'adresse i2c %11010000
; écriture de l'heure à 11:59:00 et de la date au jeudi 17/10/13
; jeudi est le 05° jour de la semaine (dimanche le 01, samedi le 07)
      symbol seconde = b0;
      symbol minute = b1;
      symbol heure = b2;
      symbol jour = b3;
      symbol date = b4;
      symbol mois = b5;
      symbol annee = b6;
      symbol control = b7;
      hi2csetup i2cmaster, %11010000, i2cslow, i2cbyte ; adressage DS1307
                                                       ; mode slow en octet
      let seconde = $00   ; positionne la variable seconde à 00
      let minute = $59    ; positionne la variable minute à 59
      let heure = $11     ; positionne la variable heure à 00
      let jour = $05      ; positionne la variable jour au jeudi donc 05
      let date = $17      ; positionne la variable date à 17
      let mois = $10      ; positionne la variable mois à 10
      let annee = $13     ; positionne la variable annee à 13
      let control = $10   ; clignotement led du module horloge $10
                          ; allumage led du module horloge $80
                          ; extinction led du module horloge $00

      hi2cout 0,(seconde,minute,heure,jour,date,mois,annee,control)      ;chargement
                                                                         ;des infos dans le module horloge
      end                 ; fin de la mise à l'heure du module horloge
```

2.36 hi2csetup

76

Syntaxe:

HI2CSETUP OFF

HI2CSETUP I2CSLAVE, slaveaddress

HI2CSETUP I2CMASTER, slaveaddress, mode, addresslen

Le mode master positionne le PICAXE en mode maître ce qui lui permet de contrôler le bus i2c.

Le mode slave positionne le PICAXE en mode esclave, dans ce mode il est commandé par un autre PICAXE en mode master.

- slaveaddress est l'adresse de l'esclave i2c.
- mode permet de choisir la vitesse de bus i2c : i2cfast(400kHz) ou i2cslow(100kHz). Utiliser i2cfast_8 ou i2cslow_8 pour les PICAXEs à 8MHZ.
- addresslen indique si la donnée est byte (i2cbyte) ou word (i2cword).

Fonction:

Permet de configurer le fonctionnement du bus i2c sur les broches du PICAXE.

Information:

L'utilisation des commandes i2c est abordée en détails dans la fiche technique du tutorial i2c.

Exemple:

```
; Utilisation d'un circuit i2c pcf8574
; le PICAXE est maître, le pcf8574 est l'esclave et utilise l'adresse i2c %01000000

      symbol data = b0;
      hi2csetup i2cmaster, %01000000, i2cslow, i2cbyte ; adressage pcf8574,
                                                       ;mode slow en octet
debut:
      let data = 240      ; charge data avec la valeur 240 soit %11110000
      hi2cout (data)      ; envoi de la donnée data
      pause 500           ; attente de 0,5 seconde
      let data = 15       ; charge data avec la valeur 15 soit %00001111
      hi2cout (data)      ; envoi de la donnée data
      pause 500           ; attente de 0,5 seconde
      goto debut          ; aller à debut, bouclage du programme
```

2.37 hi2csetup - mode esclave (PICAXES X2 seulement)

76

Slave Address

L'adresse de l'esclave permet l'identification de celui-ci sur le bus i2c. C'est un nombre compris entre 1 et 127 à condition de respecter le codage sur les 7 bits de poids forts, bit7 à bit1 par exemple %1010000x .soit (160), le bit0 est utilisé pour indiquer le mode lecture/écriture et peut donc être ignoré.

Dans le cadre du protocole i2c, certaines adresses ont une signification particulière et ne sont pas recommandées car ils peuvent provoquer un comportement inattendu des appareils reliés au bus i2c.

Information:

Quand un PICAXE est en mode esclave, le PICAXE (ou un autre microcontrôleur) maître permet la lecture et l'écriture de l'esclave comme s'il s'agissait d'une mémoire EEPROM. Le transfert de donnée peut s'effectuer avec le scratchpad.

Les commandes qui désactivent les interruptions matérielles internes (par exemple SEROUT) peuvent affecter le fonctionnement. Voir l'annexe2 pour plus de détails sur les conflits possibles.

Lorsque le maître écrit dans la mémoire d'un PICAXE esclave, le « hi2cflag » est positionné et la dernière donnée écrite est positionnée dans la variable « hi2clast ». Il est possible de tester le bit « hi2cflag » ou en positionnant le setintflags, le programme du PICAXE pourra réagir en conséquence lorsqu'une opération d'écriture se produit. Attention, le hi2cflag doit être effacé par programme après utilisation.

L'exemple suivant indique comment utiliser 2 PICAXEs 28X1, l'un en maître et l'autre en esclave.

Exemple:

```
; Utilisation d'un PICAXE 28X1 utilisé en esclave
; le PICAXE esclave utilise l'adresse i2c %10100000

init:
      hi2csetup i2cslave, %10100000        ; indication de l'adresse i2c du PICAXE esclave

debut:
      if hi2cflag = 0 then debut           ; bouclage sur début tant que le flag est à 0
                                           ; permet d'attendre une information
      hi2cflag = 0                         ; raz du flag
      get hi2clast,b1                      ; dernier octet écrit dans la variable b1
      let outpins = b1                     ; positionne les pins de sortie avec la valeur de b1
      goto debut                           ; aller à debut, bouclage du programme


; Utilisation d'un PICAXE 28X1 en maître

init:
      hi2csetup i2cmaster, %10100000, i2cslow, i2cbyte ; initialisation i2c
                                                       en mode maitre
debut:
      inc b1                               ; incrémentation de la variable b1
      hi2cout 0,(b1)                       ; écriture de la donnée b1 dans l'esclave
      pause 500                            ; attente de 500ms
      goto debut                           ; aller à debut, bouclage du programme
```

2.38 hi2csetup - mode maître

78

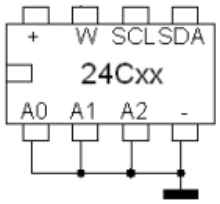
Si vous utilisez un seul périphérique i2c esclave commandé par un PICAXE maître, cela nécessite une seule commande hi2csetup dans le programme. Après l'initialisation par le hi2csetup, il est possible d'accéder à l'esclave i2c avec les commandes hi2cin et hi2cout.

Si vous utilisez plusieurs esclaves, il est possible de changer l'adresse de l'esclave dans les commandes hi2cin et hi2cout avec [newslave].

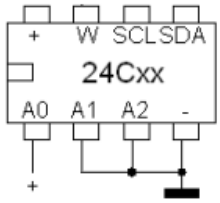
Slave Address

L'adresse de l'esclave dépend du type de périphérique i2c (voir tableau ci-dessous). Pour les EEPROM 24LCXX, l'adresse i2c est organisée par une partie fixe (bit7 à bit 4) à 1010 et par une partie configurable (bit 3 à bit1) dépendant du câblage du circuit EEPROM. Cette astuce permet de pouvoir disposer jusqu'à 8 circuits EEPROM sur le bus i2c de %10100000 à %10101110.

Le bit0 est utilisé par le protocole i2c pour indiquer s'il s'agit d'une lecture ou d'une écriture de l'esclave, cependant les commandes PICAXE readi2c ou writei2c commandent le positionnement de ce bit0.



	A6	A5	A4	A3	A2	A1	A0	bit0
bits :								
adresse i2c :	1	0	1	0	0	0	0	0



	A6	A5	A4	A3	A2	A1	A0	bit0
bits :								
adresse i2c :	1	0	1	0	0	0	1	0

Les mémoires 24Cxx sont des mémoires à lecture et écriture électrique appelé EEPROM. Elles sont prévues pour supporter 1.000.000 cycles d'écriture et conserver l'information pendant au moins 40 ans....La broche 7 : WP permet d'autoriser l'écriture sur l'EEPROM.

La plupart des fiches techniques donnent l'adresse i2c en format 8 bits, si le format est sur 7 bits, il est nécessaire d'effectuer un décalage vers la gauche pour prendre en compte le bit0 de lecture/écriture.

Vitesse du bus i2c

La vitesse du bus i2c peut être sélectionnée en sélectionnant i2cfast(400kHz) ou i2cslow(100kHz). Utiliser i2cfast_8 ou i2cslow_8 pour les PICAXEs à 8MHZ ou i2cfast_16 ou i2cslow_16 pour les PICAXEs à 16MHZ.

Il est nécessaire d'utiliser la vitesse d'horloge la plus lente des périphériques du bus i2c, par exemple, il ne faut pas utiliser i2cfast si l'un des esclaves est à 100KHz (par exemple le module horloge DS1307)

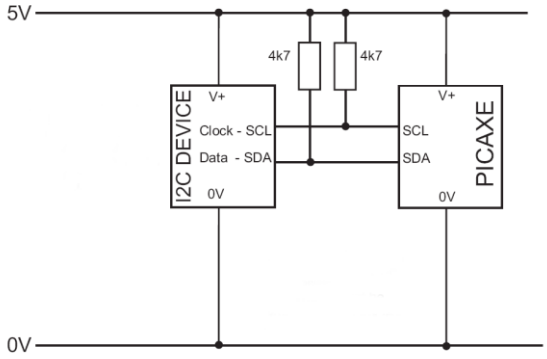
Longueur de l'adresse

Les périphériques i2c peuvent utiliser des données sur 1 octet (mode i2cbyte) ou sur 2 octets (mode i2cword). Lors de l'utilisation de i2cword veillez à bien configurer les commandes hi2cin ou hi2cout pour prendre en compte la taille des données indiquées, sans précaution, un comportement erratique interviendra.

Device	Type	Slave	Speed	Mode
24LC01B	EE 128	%1010xxxx	i2cfast	i2cbyte
24LC02B	EE256	%1010xxxx	i2cfast	i2cbyte
24LC04B	EE 512	%1010xxbx	i2cfast	i2cbyte
24LC08B	EE 1kb	%1010xbbx	i2cfast	i2cbyte
24LC16B	EE 2kb	%1010bbbx	i2cfast	i2cbyte
24LC64	EE 8kb	%1010dddx	i2cfast	i2cword
24LC128	EE 16kb	%1010dddx	i2cfast	i2cword
24LC256	EE 32kb	%1010dddx	i2cfast	i2cword
24LC512	EE 64kb	%1010dddx	i2cfast	i2cword
DS1307	RTC	%1101000x	i2cslow	i2cbyte
MAX6953	5x7 LED	%101ddddx	i2cfast	i2cbyte
AD5245	Digital Pot	%010110dx	i2cfast	i2cbyte
SRF08	Sonar	%1110000x	i2cfast	i2cbyte
AXE033	I2C LCD	$C6	i2cslow	i2cbyte
CMPS03	Compass	%1100000x	i2cfast	i2cbyte
SPE030	Speech	%1100010x	i2cfast	i2cbyte

x : sans importance b : bloc interne de sélection d : dispositif de sélection (brochage)

La présence des résistances de pull-up de 4k7 est indispensable pour le fonctionnement du bus i2c :



2.39 halt

80

Syntaxe:

HALT moteur

- Moteur est le moteur, nommé A ou B.

Fonction:

Génère un arrêt moteur

Information :

Il s'agit là d'une "pseudo commande", destinée aux étudiants les plus jeunes, dans le cadre des polycopiés de cours.

Cette commande est équivalente aux instructions 'low 4:low 5' (moteur A) ou 'low 6 : low 7' (moteur B).

Cette commande n'a, normalement, pas lieu d'être utilisée en dehors des salles de cours.

Exemple:

```
main: forward A ; moteur en avant
wait 5          ; attente 5 secondes
backward A              ; moteur en arrière
wait 5          ; attente 5 secondes
halt A          ; arrêt moteur A
wait 5          ; attente 5 secondes
goto main               ; retour début
```


(c) Revolution Education Ltd.
All rights reserved

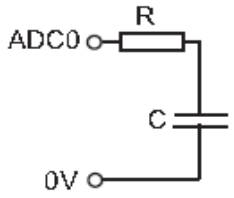
2.40 hibernate

81

Syntaxe:

HIBERNATE configuration

- Configuration est une variable, ou constante qui définit le type d'hibernation..



Fonction:

Passe le µP en mode sommeil, jusqu'à ce que se produise un reset, ou une interruption.

Information :

La commande hibernate force le µP en mode très basse consommation,. Contrairement à la commande Sleep, qui teste l'état de sommeil toutes les 2.3 s, le mode hibernate passe le µP en sommeil permanent. Le seul moyen de sortir le µP de ce sommeil profond est d'utiliser un reset extérieur (hard reset), ou une interruption matérielle (hserin, hi2cin, etc ...).

Une nouvelle reprogrammation, via câble et PC ne réveillera PAS le µP.

Afin d'obtenir la consommation la plus réduite possible, assurez vous de maintenir a un état déterminé (haut ou bas) les différents ports d'entrée-sortie (rien "en l'air"), et qu'aucune des sortie ne soit connectée activement à une charge. La commande hibernate inhibe automatiquement tous les périphériques embarqués (timers, pwm, etc...) et inhibe la détection de variation de tension d'alimentation, tout comme la commande 'disable bod'(inhibition du Brown Out Detect). Après une commande 'hibernate', la détection BOD est toujours ré-activée, et donc, si cette fonction n'est pas souhaitée, après un hibernate, il est nécessaire de la désactiver de nouveau par une commande 'disablebod'.

La valeur configuration est utilisée pour déclencher/arrêter le mode 'réveil d'un mode consommation ultra-basse' d'une entrée ADC0.La valeur 0 inhibe cette fonction. Sinon, la commande hibernate se terminera dès la fin de la décharge d'une capacité connectée en ADC0. Plus efficace, en terme de basse consommation, que la commande sleep.

Une valeur différente de 0 valide la fonction ULPWU (Utra Low Power Unit) sur le port ADC0, et active les paramètres de configuration du temps de charge de la capacité (en ms). Toutefois, la commande hibernate charge d'abord la capacité , puis passe en mode sommeil, et repasse en mode éveil, une fois la capacité déchargée.

Le temps de décharge de la capacité est donné par la formule suivante :

Temps = ((Tension initiale aux bornes de la capacité – 0.6 * C) / (courant de décharge + courant de fuite)

Le courant de décharge est approximativement de 140 nA pour une tension d'alimentation de 5 volts. Ainsi, le temps de décharge, pour une capacité de 1nF, connectée sur une résistance de 200 ohms est d'environ 30 mS. Dans ces conditions, la commande hibernate se terminera au bout d'environ 30 mS, bien que le temps de décharge soit très dépendant de la valeur de la capacité (du condensateur, ET du circuit associé), donc, la longueur des piste du circuit imprimé, et leur état de salissure sont susceptibles d'influer considérablement sur le temps de réponse.

2.41 high

83

Syntaxe:

HIGH broche {,broche,broche,...}

"broche" est une variable ou un constante qui spécifie la broche d'entrée/sortie à utiliser.

Si besoin, " broche" désigne et le port et la broche (voir l' exemple ci-dessous)

Concerne :

08 / 08M / 08M2

14M / 14M2

18 / 18A / 18M / 18M2 / 18X

20M / 20M2 / 20X2

28A / 28X / 28X1 / 28X2

40X / 40X1 / 40X2

Commentaire :

L' instruction "high" force une sortie au niveau haut.

Sur les modèles où les broches sont configurables en entrée ou en sortie (par exemple PICAXE08) cette commande configure automtiquement la broche en sortie (et au niveau haut).

Exemple:

```
high B.1      ; place la broche 1 du port B en sortie et au niveau haut
```


(c) Revolution Education Ltd.
All rights reserved

2.42 high portc

Syntaxe:

HIGH PORTC brohe {,broche,broche,...}

"broche" est une variable ou une constante qui spécifie la broche d'entrée/sortie du port C à utiliser.

Concerne :

Cette instruction ne concerne que les anciens modèles 14M et 28X / 28X1.

Pour les modèles M2 et X2 utiliser directement la notation "port.broche", par exemple " high C.2 ".

Commentaire :

Place la broche désignée au niveau haut.

Exemple:

```
high portc 1 ; place la broche 1 du port C au niveau haut.
```

2.43 hintsetup

85

Syntaxe:

HINTSETUP masque

- Masque est une variable, ou constante, qui définit quelle broche activera l'interruption.

  Bit 7 -réservé

  Bit 6 -Interruption 2 Trigger (1=flanc montant, 0=flanc descendant)

  Bit 5 -Interruption 1 Trigger (1=flanc montant, 0=flanc descendant)

  Bit 4 -Interruption 0 Trigger (1=flanc montant, 0=flanc descendant)

  Bit 3 -*Réservé*

  Bit 2 -Interruption 2 validée

  Bit 1 -Interruption 1 validée

  Bit 0 -Interruption 0 validée (sauf sur la puce 20X2)

Fonction:

Les puces type X2 disposent de trois entrées d'interruption (INT0, INT1, INT2) qui peuvent être activées/désactivées via la commande hintsetup. Cette commande va vérifier en permanence l'arrivée d'un front (changement brutal d'état) sur ces broches. Cette action se déroulant en tâche de fond, le programme principal n'a pas besoin de vérifier régulièrement l'état de ces entrées.

Le traitement de ces interruptions est extrêmement rapide. Soyez donc prudent, lors de leur utilisation, quant aux éventuels phénomènes parasites (rebond de bouton poussoir par exemple, qui pourraient donner un résultat imprévisible).

Une interruption peut également générer un éveil d'un PICAXE , préalablement passé en mode sommeil.

Information :

Une interruption va causer un changement immédiat de drapeau d'état de la broche concernée. Si la commande setinflags a également été activée, une interruption programme PICAXE peur être générée.

L'activation individuelle de chaque broche positionne deux drapeaux, l'un, unique, et le second, dit 'hintflag' partagé. Ces drapeaux peuvent être manuellement effacés dans le programme utilisateur. La commande hintsetup ne valide que les entrées concernées par ces drapeaux, sans modifier une interruption prévue dans le programme PICAXE.

Donc, afin que le programme PICAXE appelle une interruption, fonction de l'état d'une broche d'entrée, vous devez procéder comme suit :

Utiliser 'hintsetup' afin de positionner le drapeau matériel.

Puis, utiliser 'setintflags' afin de générer une interruption en cas de choix de ce drapeau. Cela signifie, qu'il est possible d'obtenir une interruption avec soit un des drapeaux, soit de tous les drapeaux, grâce à cette commande. Voir le paragraphe 'setinflags' pour plus de détails.

Exemple:

```
hintsetup %00000111     ; valide les  3 entrées
hintsetup %00000010     ; ne valise que INT1
hintsetup %00000000     ; invalide tout
```


(c) Revolution Education Ltd.
All rights reserved

2.44 hpwm

86

Syntaxe:

HPWM mode, polarité, positionnement, période, rapport cyclique

HPWM DIV4, polarité, positionnement, période, rapport cyclique

HPWM DIV16, polarité, positionnement, période, rapport cyclique

HPWM DIV64, polarité, positionnement, période, rapport cyclique

HPWM OFF

- Mode est une variable, ou constante, qui définit le mode pwm matériel.
  - Pwmsingle - 0
  - Pwmhalf - 1
  - Pwmfull_f - 2
  - Pwmfull_r - 3
- Polarité est une variable, ou constante, qui définit la polarité active (DCBA)
  - PwmHHHH - 0
  - PwmLHLH - 1
  - PwmHLHL - 2
  - PwmLLLL - 3
- Positionnement est une variable, ou constante, qui spécifie un positionnement
  - Mode simple - masque bit de %0000 à %1111 pour activer,/désactiver DCBA
  - Demi mode - durée du temps mort (valeur de 0 à 127)
  - Mode plein - inutilisé, la valeur par défaut est 0
- Période est une variable, ou une constante (0 à 255) qui définit la période du pwm

  (la période est la longueur d'un cycle on/off, fonction de la longueur totale du nombre de cycles)
- Rapport cyclique est une variable, ou constante (0-1023) qui définit le rapport cyclique du pwm. Le rapport cyclique est définit par le ratio état haut/état bas sur une durée donnée.

Le mot clé PWMDIV est utilisé pour diviser la fréquence du signal pwm par 4, 16, ou 64. La valeur 64 n'et pas supportée par toutes les puces.

Il est a noter, que le menu d'aide du programme consacré au PICAXE comporte une rubrique permettant de calculer aisément les paramètres de la génération d'un pwm. (PICAXE→WIZARD→PWM). Voir la commande pwmout pour plus de détails.

Puces type DIL28 -les 28X1, 28X2, 28X2-3V supportent hpwm, PAS les 28X2-5V.

Puces type DIL40 -les 40X2, 40X2-5V et 40X2-3V supportent hpwm, PAS les 40X1

Ces restrictions sont dues à la conception physique des puces.

Fonction:

La génération matérielle d'un signal pwm est une méthode moderne permettant le contrôle de vitesse de rotation de moteurs. Il est ici possible de générer simultanément plusieurs signaux de commande selon le paramétrage adopté pour le µP.

Hpwm peut être utilisé à la place de pwmout, mais PAS en même temps que la commande pwmout on 2 (28/40 broches). Cependant, pwmout on 1 peut-être utilisé en même temps, si on le souhaite.

Description :

Hpwm donne directement accès au contrôleur pwm implémenté dans le PIC. Cela utilise 4 broches, dénommées A, B, C, D par commodité. Certaines de ces broches sont par défaut, paramétrées en entrées. Elle seront automatiquement re-paramétrées en sortie, lors de l'utilisation de hpwm.

Sur les puces DIL20:	Sur les puces DIL14:
A est l'entrée 5 (C.5)	A est l'entrée 2 (C.5)
B est l'entrée 4 (C.4)	B est l'entrée 1 (C.4)
C est l'entrée 3 (C.3)	C est l'entrée 0 (C.3)
D est la sortie 4 (B.4)	D est la sortie 5 (C.2)

Sur les puces DIL28	Sur les puces DIL40
A est l'entrée 2 (C.2)	A est le port C2 (C.2)
B est la sortie 2 (B.2)	B est l'entrée 5 (D.5)
C est la sortie 1 (B.1)	C est l'entrée 6 (D.6)
D est la sortie 4 (B.4)	D est l'entrée 7 (D.7)

Toutes les broches ne peuvent pas être utilisées avec hpwm. Les inutilisées demeurent de simples entrées/sorties.

Simple -A et/ou B et/ou C et/ou D (chaque bit est sélectionnable)

Demi -A, B seulement

Complet -A, B, C, D

La polarité active de chaque paire de broches peut-être définite par 'polarité'

Pwm_HHHH	-A et C actifs haut, B et D actifs haut
Pwm_LHLH	-A et C actifs haut, B et D actifs bas
Pwm_HLHL	-A et C actifs bas, B et D actifs haut
Pwm_LLLL	-A et C actifs bas, B et D actifs bas

Lors de l'utilisation de sorties actives à l'état haut, il est important d'utiliser des résistances de forçage à la masse (A,D vers 0V). De même, pour des sorties active ç l'état bas, des résistance de forçage au +V sont nécessaires.

La raison d'être de ces résistances est de maintenir les drivers type FET, présent dans le µP a un potentiel correct, durant les phases d'initialisation, lors de la mise sous tension.

Mode simple

Supporté: 20X2, 28X1, 28X2, 28X2-3V, 40X2, 40X2-3V

Non supporté: 14M, 14M2, 20M2, 28X2-5V, 40X1, 40X2-5V

En mode simple, chaque broche travaille indépendamment. C'est donc l'équivalent d'une commande pwmout. Il est cependant possible d'activer plusieurs broches simultanément.

Ce mode a donc deux usages principaux :

Autoriser du pwmout sur plusieurs broches

Autoriser la simultanéité (jusque 4 sorties en même temps). Les signaus générés simultanément sont identiques. Cette technique est fréquemment utilisée pour du contrôle de luminosité de panneaux de LED, ou contrôle de vitesse de moteurs.

Afin d'utiliser une seule sortie, il suffit de positionner le bit correspondant à '1' (D-C-B-A) dans la séquence byte de commande, par exemple, pour activer les 4 broche, utilisez %1111

Demi mode (toutes puces)

En demi mode, les sortie A et C contrôlent un demi pont. C et D ne sont pas utilisées. Le signal PWM est généré sur la broche A, son complément sur la broche B. Le paramétrage de temps mort est très important, sans une valeur correcte de ce dernier, le pont commandé risque de se trouver dans un état “double passant”, entrainant sa destruction.

La valeur du temps mort (0-127) est déterminée comme suit : valeur X vitesse horloge / 4 .

Cette valeur doit être adaptée aux caractéristiques de vitesse de basculement des FET utilisés dans le pont.

Voir les feuilles de caractéristiques des ponts utilisés pour plus de détails.

Mode total (toutes puces)

En mode total, les sorties A, B, C, D contrôlent le pont.

En marche avant, A est à l’état haut, pendant que D est modulé. B et C sont inactifs.

Dans l’autre sens, C est à l’état haut, pendant que B est modulé. A et D sont inactifs.

Dans ce mode, un temps mort n’est généralement pas nécessaire, dans la mesure ou une seule sortie est modulée à la fois.

Néanmoins, il peut y avoir des cas (100% de rapport cyclique) ou le risque demeure.

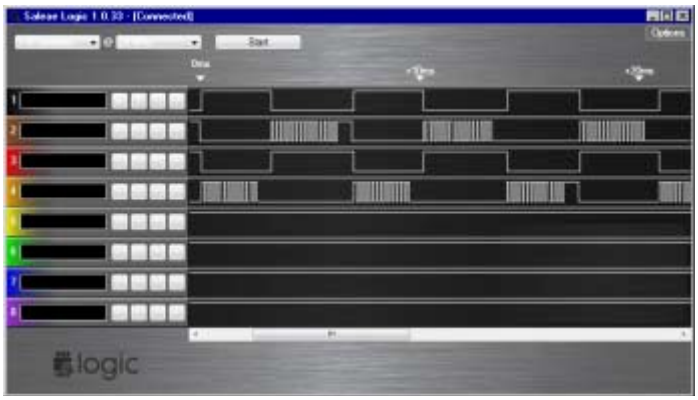
Dans ce cas :

- coupez la commande pwm avant de changer de sens de rotation
- utilisez un pont de FET qui bascule plus vite en mode ON qu’en mode OFF (en général, les composants non fait pour cela font plutôt l’inverse)

Voir les feuilles de caractéristiques des ponts utilisés pour plus de détails.

Hpwm Mode simple

Hpwm Mode total



2.45 hpwmduty

90

Syntaxe:

HPWMDUTY rapport cyclique

- Rapport cyclique est une variable, ou constante, qui définit le rapport cyclique du pwm . Le rapport cyclique est définit par le ratio état haut/état bas sur une durée donnée.

.

Fonction:

Modifie le rapport cyclique après application de la commande.

Information :

La commande hpwmduty peut être utilisée pour modifier le rapport cyclique de la commande hpwm sans devoir remettre à zéro le compteur interne.(ce qui se produit, avec une commande hpwm).

Une commande hpwm doit être utilisée avant d'utiliser hpwmduty

Information:

Voir **hpwm** pour plus de détails.

Exemple:

```
init:
        hpwm 0,0,%1111,150,100          ; démarre pwm

main:
        hpwmduty 150                    ; fixe le rapport cyclique
        pause 1000                      ; attente  1 s
        hpwmduty 50                     ; modifie le rapport cyclique
        pause 1000                      ; attente 1 s
        goto main                       ; retour au début
```


(c) Revolution Education Ltd.
All rights reserved

2.46 hserin

91

Syntaxe (puces X2):

HSERIN spadresse, compte {,(qualifieur)}

HSERIN [timeout, adresse],spadresse, compte {,(qualifieur)}

- Qualifieur est une variable ou constante optionnelle qui doit être reçue avant la réception et la mémorisation de la suite de la réception, dans le scratchpad. Le scratchpad est une zone mémoire utilisée pour stocker des données.
- Spadresse est la première adresse dans le scratchpad, ou seront stockés les bytes reçus
- Compte est le nombre de bytes a recevoir
- Timeout est une variable ou constante optionnelle qui détermine le délais d'attente en millisecondes
- Adresse est l'adresse ou le programme doit poursuivre, en cas de dépassement du délais précité.
-

Syntaxe (puces M2):

HSERIN variable

- *Variable est la variable qui stocke le byte reçu*

Fonction:

Entrée série, depuis le port entrée série (format : 8 bits de données, pas de parité, 1 bit stop).

Information :

La commande hserin est utilisée pour recevoir des données séries, via l'entrée série fixée par le matériel. Elle ne peut généralement pas être utilisée avec l'entrée série de programmation –dans ce cas, utilisez l'instruction serrxd

La vitesse de transmission est fixée par l'instruction hsersetup qui doit être utilisée au préalable.

Les utilisateurs familiers de la commande serin remarqueront que le format de hserin est totalement différent. Ceci est dû au fait que la commande hserin supporte des vitesses de transmission beaucoup plus élevées que la commande serin, et est donc incapable de saisir et traiter "au vol" des données entrantes (par exemple, conversion ASCII binaire, comme faisable avec l'utilisation du préfixe '#'). A ces vitesses, le temps de traitement est trop court, pour ne pas 'rater' l'arrivée du byte suivant, avant d'avoir fini de traiter la donnée précédente. En conséquence, les données sont simplement sauvegardées en mémoire, jusqu'à la fin de transmission. Dès lors, elles pourront être traitées par le programme.

***Exemple (puces X2)*:**
Notez qu'avec les puces de type X2, vous pouvez privilégier la réception de données en arrière plan, avec stockage en scratchpad (cette méthode ne nécessite absolument pas l'utilisation de cette commande) voir l'instruction hersetup pour plus de détails (hserin accepte seulement les données, durant l'exécution de la commande, la réception en arrière plan les accepte à tout moment

```
hsersetup B19200_16, %00                ; 19200 bauds à 16MHz
main:
        hserin [1000,main],0,4          ; réception 4 bytes en scratchpad
        ptr = 0                         ; remise à 0 du pointeur scratchpad
        hserout 0,(@ptrinc,@ptrinc,@ptrinc,@ptr)   ; renvoi écho
        goto main                       ; retour début
```

***Exemple (puces M2)*:**

Pour les puces M2, la commande hserin est utilisée pour transférer dans une variable des données reçues en arrière plan. Jusque deux bytes peuvent être reçus en arrière plan à n'importe que moment du déroulement du programme (et non pas juste au moment de l'exécution de hserin). Ces données sont stockées dans une mémoire FIFO à deux niveaux (FIFO: First In First Out, la première donnée stockée ressortira la première, lors de la relecture).
Au delà de ces deux bytes, toute autre données reçue sera perdue.

Cependant, sur les puces M2, la commande hserin n'est pas bloquante, elle s'exécute toujours immédiatement. Si des données entrantes sont présentes dans le tampon, le premier des bytes est transféré dans la variable, sinon, la variable n'est pas modifiée, et le programme se poursuit à la ligne suivante. Si deux bytes sont attendus, il est nécessaire d'utiliser deux commandes hserin, pour récupérer ces deux bytes.

```
hsersetup B9600_4, %00                  ; 9600 bauds à 4MHz
main:
        w1 = $FFFF                      ; Définit une variable invalide
        hserin w1                       ; réception 1 byte dans w1
        if w1 <> $FFFF then             ; test si réception un byte
                hserout 0,(w1)          ; echo renvoyé
        end if
        goto main                       ; retour début
```


(c) Revolution Education Ltd.
All rights reserved

2.47 hserout

93

Syntaxe :

HSEROUT break, ({#}donnée,{#}donnée...)

- Break est une variable ou constante (0 ou 1) permettant d'informer le système destinataire de l'envoi d'une donnée (réveil)
- Data est une variable, ou constante (0 à 255) qui spécifie la donnée a émettre. Le #, optionnel, permet de convertir la donnée en valeur ASCII décimale, plutôt qu'en caractère brut. Les données de type texte doivent être placées entre guillemet ("Bonjour")

Fonction:

Envoie d'une donnée série, depuis le port sortie série (format : 8 bits de données, pas de parité, 1 bit stop).

Information :

La commande hserout est utilisée pour envoyer des données séries, via la sortie série fixée par le matériel. Elle ne peut pas être utilisée avec le port série de programmation –dans ce cas, utilisez l'instruction serrxd

La vitesse de transmission et la polarité doivent être préalablement fixées par l'instruction hsersetup

Le symbole # force l'envoi en mode ASCII. Ainsi, l'envoi de la donnée #b1, avec b1=126 enverra les caractères "1" "2" "6" en lieu et place de '126'

Exemple :

```
hsersetup B2400_4, %10              ; 2400 baud, polarité inverse
main:
        for b0 = 0 to 63            ; départ boucle
                read b0,b1          ; lit la valeur, et la place dans b1
                hserout 0,(b1)      ; transmet la valeur au LCD série
        next b0                     ; valeur suivante
```


(c) Revolution Education Ltd.
All rights reserved

2.48 hsersetup

94

Syntaxe :

HSERSETUP OFF

HSERSETUP choix_vitesse, choix mode

- Choix vitesse est une variable ou constante qui définit la vitesse de transmission :
  - B300_X Ou X=
  - B600_X 4 pour 4Mhz
  - B1200_X 8 pour 8Mhz
  - B2400_X 16 pour 16Mhz
  - B4800_X 20 pour 20Mhz
  - B9600_X 32 pour 32Mhz
  - B19200_X 40 pour 40Mhz
  - B31250_X 64 pour 64Mhz
  - B38400_X
  - B57600_X
  - B_115200_X

Choix mode est une variable, ou constante qui spécifie certaines fonctions spéciales (pas implémenté sur toutes les puces) :

- Bit0 – réception en tâche de fond dans le scratchpad (pas avec les puces M2)
- Bit1 – inversion des données émises (0='T', 1='N')
- Bit2 – inversion des données reçues (0='T', 1='N')
- Bit3 – désactive hserout (1 = broche hserout mode normal in/out)
- Bit4 – désactive hserin (1 = broche hserin mode normal in/out)

Fonction:

Configuration du port série

Information :

La commande hsersetup est utilisée pour configurer le port série du µP. Cette configuration affecte les deux broches du port (entrée/sortie), sans possibilité de n'en affecter qu'une seule.

La vitesse de transmission est fixée par le paramètre choix_vitesse. Cette valeur fixe le débit. Par commodité, des valeurs pré-définites peuvent être utilisées. Ainsi, B9600_4 correspond à une vitesse de 9600 bauds, pas de parité, huit bits de données et1 bit stop, à 4Mhz de fréquence d'horloge.

Il est cependant toujours possible de fixer d'autres paramètres, en utilisant la formule exposée plus loin.

Le port de réception série peut-être configuré de deux manières :

- via la commande hserin uniquement (mode bit0=0)
- automatiquement en mode tâche de fond (mode bit0=1) (PAS sur les puces type M2)

En mode réception automatique en tâche de fond, les paramètres du port entrée série sont totalement automatisés. Le flux de données séries reçu est directement sauvegardé dans le scratchpad. Prioritairement sur la commande hsetup, le pointeur série (hserptr) est remis à 0. Lors d'une réception de donnée, celle-ci est transférée à cette adresse du scratchpad, la variable hserptr est incrémentée, et le drapeau hserinflag est activé (il devra être manuellement désactivé par le programme utilisateur).

De ce fait, la valeur 'hserptr – 1' spécifie le dernier byte écrit, et la valeur 'hserinflag=1' spécifie qu'un byte vient d'être reçu (voir également la commande setinflag).

Le scratchpad est une mémoire circulaire, qui peut passer en dépassement sans prévenir.

Polarité :

Lorsque que le bit 1 est à 0, la polarité de la sortie série est dite 'vraie' ce qui équivaut à un 'Txxx' vitesse de transmission dans la commande serout. Dans ce cas, la sortie du port est à l'état haut en mode veille. Cette configuration est standard, en cas d'utilisation d'un inverseur/adaptateur tel que le MAX232 pour une connexion avec un PC.

Lorsque le bit 1 est à 1, la polarité de la sortie série est dite 'inverse' ce qui équivaut à un 'Nxxx' vitesse de transmission dans la commande serout.. Dans ce cas, la sortie du port est à l'état bas en mode veille. Cette configuration est standard, en cas d'utilisation d'un circuit d'interface (tel que le AXE033 LCD), ou d'une connexion directe avec un PC, via des résistances pour abaisser le tension à un seuil acceptable pour le PICAXE.

Sur certains circuits, la polarité du port d'entrée est toujours 'vraie', sans possibilité d'inversion (par exemple, l'inversion du bit 2 ne s'applique qu'aux puces types X2). Ceci est dû à une limitation interne de la structure du contrôleur série. Dans ce cas précis, l'usage d'un MAX232 s'impose.

Exemple :

```
hsersetup B9600_4, %10                  ; 9600 baud, inversé TXD
main:
        for b0 = 0 to 63                ; démarrage boucle
                read b0,b1              ; lecture valeur et transfer en b1
                hserout 0,(b1)          ; écriture de la valeur vers LCD série
        next b0                         ; suite boucle
```

Information techniques complémentaires :

Il est possible de créer ses propres paramètres de configuration pour une vitesse de transfert donnée.

Le paramètre 'choix_vitesse' doit être une variable type word et peut être calculé avec l'équation suivante (ou, 'n' est la valeur vitesse) :

Vitesse souhaitée = Fréquence horloge / (4(n+1))

D'où l'on déduit :

N = ((fréquence horloge / vitesse) / 4)-1

Ainsi, pour une fréquence d'horloge de 4Mhz, on désire une vitesse de 10400 bauds, n vaudra

N = ((4 000 000 / 10400) / 4) –1 = 95 (valeur arrondie)

Une technique de vérification :

Vitesse = 4 000 000 / (4(95+1)) = 10416 (pour 10400 souhaité)

Cette marge d'erreur est tolérable par la plupart des applications.

2.49 hspiin (hshin)

96

Syntaxe :

HSPIIN (variable, {,variable,...})

- Variable reçoit la donnée

Fonction:

La commande hspiin (hshin est aussi acceptée par le compilateur) reçoit séquentiellement une donnée byte, via le port SPI.

Description :

Cette commande reçoit des données SPI, via le port dédié SPI du µP. Cette méthode est plus rapide et plus efficiente en terme de code, que l'utilisation de la méthode bit-bang utilisée par la commande spiin.

Note du traducteur :

La méthode bit-bang consiste à recueillir "à la volée", donc par micro-logiciel, l'état de chaque bit envoyé par l'émetteur. L'utilisation des circuits électroniques dédiés contenus dans certains PICs est évidement bien plus performante.

Lors de la connexion d'un composant SPI (EEPROM, par exemple) , il faut que la sortie de lecture EEPROM soit connectée à l'entrée SPI du µP, et l'entré d'écriture, connectée à la sortie SPI du µP.

Attention : la commande hspisetup doit être utilisée avant toute utilisation de hspiinn.

Exemple :

Voir la commande **hpisetup** pour un exemple détaillé.

(c) Revolution Education Ltd.
All rights reserved

2.50 hspiout (hshout)

97

Syntaxe :

HSPIOUT(donnée, {,donnée ,...})

- - donnée est une constante ou variable qui contient la valeur à emettre.

Fonction:

La commande hspiin (hshin est aussi acceptée par le compilateur) reçoit séquentiellement une donnée byte, via le port SPI.

Description :

Cette commande reçoit des données SPI, via le port dédié SPI du µP. Cette méthode est plus rapide et plus efficiente en terme de code, que l'utilisation la méthode bit-bang utilisée par la commande spiout.

Lors de la connexion d'un composant SPI (EEPROM, par exemple) , il faut que la sortie de lecture EEPROM soit connectée à l'entrée SPI du µP, et l'entré d'écriture, connectée à la sortie SPI du µP.

Note du traducteur :

La méthode bit-bang consiste à émettre "à la volée", donc par micro-logiciel, l'état de chaque bit. L'utilisation des circuits électroniques dédiés contenus dans certains PICs est évidement bien plus performante.

Exemple :

Voir la commande **hpisetup** pour un exemple détaillé.

2.51 hspisetup

98

Syntaxe :

HSPISETUP OFF

HSPISETUP mode,vitesse_spi

- Mode est une variable, ou une constante qui définit le mode :

Spimode00	(mode 0,0 – l'entrée est échantillonnée au milieu de la durée de la donnée)
Spimode01	(mode 0,1 – l'entrée est échantillonnée au milieu de la durée de la donnée)
Spimode10	(mode 1,0 – l'entrée est échantillonnée au milieu de la durée de la donnée)
Spimode11	(mode 1,1 – l'entrée est échantillonnée au milieu de la durée de la donnée)
Spimode00e	(mode 0,0 – l'entrée est échantillonnée à la fin de la durée de la donnée)
Spimode01e	(mode 0,1 – l'entrée est échantillonnée à la fin de la durée de la donnée)
Spimode10e	(mode 1,0 – l'entrée est échantillonnée à la fin de la durée de la donnée)
Spimode11e	(mode 1,1 – l'entrée est échantillonnée à la fin de la durée de la donnée)

- Vitesse_spi est une variable, ou une constante qui définit la vitesse d'horloge.

Spifast	(fréquence d'horloge / 4)	(=1 Mhz pour une horloge à 4 Mhz)
Spimedium	(fréquence d'horloge / 16)	(=250 Khz pour une horloge à 4 Mhz)
Spislow	(fréquence d'horloge / 64)	(=63 Khz pour une horloge à 4 Mhz)

Fonction:

La commande hspisetup paramètre le fonctionnement d'une broche en mode SPI.

Description :

Cette commande reçoit des données SPI, via le port dédié SPI du µP. Cette méthode est plus rapide et plus efficiente en terme de code, que l'utilisation que l'utilisation la méthode bit-bang utilisée par la commande spiout (shiftout)

Lors de la connexion d'un composant SPI (EEPROM, par exemple) , il faut que la sortie de lecture EEPROM soit connectée à l'entrée SPI du µP, et l'entré d'écriture, connectée à la sortie SPI du µP.

Note du traducteur :

La méthode bit-bang consiste à émettre "à la volée", donc par micro-logiciel, l'état de chaque bit. L'utilisation des circuits électroniques dédiés contenus dans certains PICs est évidement bien plus performante.

Information plus technique:

Les utilisateurs familiers de la programmation en mode assembleur trouveront cette méthode d'utilisation du µP plus pratique.(voir copie d'écran d'analyseur logique)

Spinmode00	(CKP=0, CKE=1, SMP=0)	Mode(0,0)
Spinmode01	(CKP=0, CKE=0, SMP=0)	Mode(0,1)
Spinmode10	(CKP=1, CKE=1, SMP=0)	Mode(1,0)
Spinmode11	(CKP=1, CKE=0, SMP=0)	Mode(1,1)
Spinmode00e	(CKP=0, CKE=1, SMP=1)	

Spinmode01e	(CKP=0, CKE=0, SMP=1)
Spinmode10e	(CKP=1, CKE=1, SMP=1)
Spinmode11e	(CKP=1, CKE=0, SMP=1)

Exemple :

Cet exemple montre comment lire et écrire dans une EEPROM 25LC160.

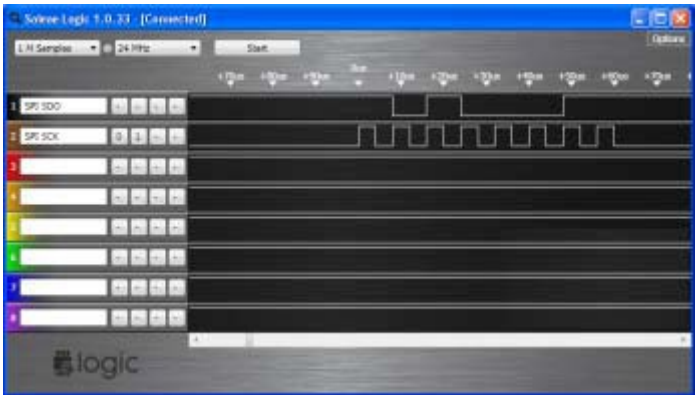
Le connections a l'EEPROM sont établies de la manière suivante :

1 – CS	PICAXE sortie 7 (B.7)
2 – SO	PICAXE entrée 4 (C.4)
3 – WP	+5V
4 – Vss	0V
5 – SI	PICAXE entrée 5 (C.5)
6 – SCK	PICAXE entrée 3 (C.3)
7 – HOLD	+5V
8 + Vdd	+5V

```
init:
    hspisetup spimode11e, spimedium     ; spi mode 1,1
    low cs                              ; valide choix composant
    hspiout (6)                         ; valide l'envoi en mode écriture
    high cs                             ; invalide choix composant
    low cs                              ; valide choix composant
    hspiout (1,0)                       ; ote la protection block
    high cs                             ; invalide choix composant
    pause 5                             ; attente temps écriture

main:
    low cs                              ; valide choix composant
    hspiout (6)                         ; valide l'envoi mode écriture
    high cs                             ; invalide choix composant
    low cs                              ; valide choix composant
    hspiout (2,0,5,25)                  ; ecrit  25 à l'adresse 5
    high cs                             ; invalide choix composant
    pause 5                             ; attente temps écriture 5ms
    low cs                              ; valide choix composant
    hspiout (6)                         ; valide envoi mode écriture
    high cs                             ; invalide choix composant
    low cs                              ; valide choix composant
    hspiout (3,0,5)                     ; envoie commande de lecture,adresse 5
    hspiin (b1)                         ; décale d'un cran la donnée
    high cs                             ; invalide choix composant
    low cs                              ; valide choix composant
    hspiout (4)                         ; invalide le mode écriture
    high cs                             ; invalide choix composant
    debug                               ; contrôle process sur PC
    pause 1000                          ; attente 1S
    goto main                           ; retour début
```

Hspiout – mode

hspiout – mode 01



(c) Revolution Education Ltd.
All rights reserved

2.52 i2cslave

102

Cette commande est obsolète, pensez à utiliser la commande hi2csetup.

Syntaxe:

I2CSLAVE, slaveaddress, mode, adresslen

- slaveaddress est l'adresse de l'esclave i2c.
- mode permet de choisir la vitesse de bus i2c : i2cfast(400kHz) ou i2cslow(100kHz).
- addresslen indique si la donnée est byte (i2cbyte) ou word (i2cword).

Fonction:

Permet de configurer les broches du PICAXE pour en mode i2cmaster.

Information:

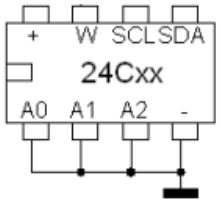
L'utilisation des commandes i2c est abordée en détails dans la fiche technique du tutorial i2c.

Si vous utilisez un seul périphérique i2c, une seule commande i2cslave suffit. Avec le PICAXE 18X, vous devez utiliser cette commande en début de programme pour configurer les broches SDA et SCL. Après cette commande, il est possible d'utiliser readi2c ou writei2c pour accéder au périphérique i2c.

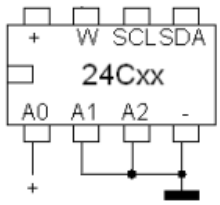
Slave Address

L'adresse de l'esclave dépend du type de périphérique i2c (voir tableau ci-dessous). Pour les EEPROM 24LCXX, l'adresse i2c est organisée par une partie fixe (bit7 à bit 4) à 1010 et par une partie configurable (bit 3 à bit1) dépendant du câblage du circuit EEPROM. Cette astuce permet de pouvoir disposer jusqu'à 8 circuits EEPROM sur le bus i2c de %10100000 à %10101110.

Le bit0 est utilisé par le protocole i2c pour indiquer s'il s'agit d'une lecture ou d'une écriture de l'esclave, cependant les commandes PICAXE readi2c ou writei2c commandent le positionnement de ce bit0.



	A6	A5	A4	A3	A2	A1	A0	bit0
bits :								
adresse i2c :	1	0	1	0	0	0	0	0



	A6	A5	A4	A3	A2	A1	A0	bit0
bits :								
adresse i2c :	1	0	1	0	0	0	1	0

Les mémoires 24Cxx sont des mémoires à lecture et écriture électrique appelé EEPROM. Elles sont prévues pour supporter 1.000.000 cycles d'écriture et conserver l'information pendant au moins 40 ans....La broche 7 : WP permet d'autoriser l'écriture sur l'EEPROM.

La plupart des fiches techniques donnent l'adresse i2c en format 8 bits, si le format est sur 7 bits, il est nécessaire d'effectuer un décalage vers la gauche pour prendre en compte le bit0 de lecture/écriture.

Vitesse du bus i2c

La vitesse du bus i2c peut être sélectionnée en sélectionnant i2cfast(400kHz) ou i2cslow(100kHz). Utiliser i2cfast_8 ou i2cslow_8 pour les PICAXEs à 8MHZ ou i2cfast_16 ou i2cslow_16 pour les PICAXEs à 16MHZ.

Il est nécessaire d'utiliser la vitesse d'horloge la plus lente des périphériques du bus i2c, par exemple, il ne faut pas utiliser i2cfast si l'un des esclaves est à 100KHz (par exemple le module horloge DS1307)

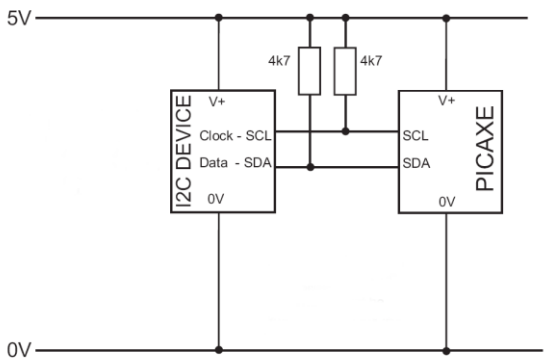
Longueur de l'adresse

Les périphériques i2c peuvent utilisés des données sur 1 octet (mode i2cbyte) ou sur 2 octets (mode i2cword). Lors de l'utilisation de i2cword veillez à bien configurer les commandes hi2cin ou hi2cout pour prendre en compte la taille des données indiquées, sans précaution, un comportement erratique interviendra.

Device	Type	Slave	Speed	Mode
24LC01B	EE 128	%1010xxxx	i2cfast	i2cbyte
24LC02B	EE256	%1010xxxx	i2cfast	i2cbyte
24LC04B	EE 512	%1010xxbx	i2cfast	i2cbyte
24LC08B	EE 1kb	%1010xbbx	i2cfast	i2cbyte
24LC16B	EE 2kb	%1010bbbx	i2cfast	i2cbyte
24LC64	EE 8kb	%1010dddx	i2cfast	i2cword
24LC128	EE 16kb	%1010dddx	i2cfast	i2cword
24LC256	EE 32kb	%1010dddx	i2cfast	i2cword
24LC512	EE 64kb	%1010dddx	i2cfast	i2cword
DS1307	RTC	%1101000x	i2cslow	i2cbyte
MAX6953	5x7 LED	%101ddddx	i2cfast	i2cbyte
AD5245	Digital Pot	%010110dx	i2cfast	i2cbyte
SRF08	Sonar	%1110000x	i2cfast	i2cbyte
AXE033	I2C LCD	$C6	i2cslow	i2cbyte
CMPS03	Compass	%1100000x	i2cfast	i2cbyte
SPE030	Speech	%1100010x	i2cfast	i2cbyte

x : sans importance b : bloc interne de sélection d : dispositif de sélection (brochage)

La présence des résistances de pull-up de 4k7 est indispensable pour le fonctionnement du bus i2c :



2.53 if...then \elseif...then\else\endif

104

Syntaxe :

```
IF valeur variable= ?? valeur {AND/OR variable ?? valeur} THEN
  {code}
ELSEIF variable ?? valeur {AND/OR variable ?? valeur...} THEN
  {code}
ELSE
  {code}
ENDIF
```

Options supplémentaires, uniquement pour les puces X1 et X2 :

```
IF variable BIT valeur SET THEN
  {code}
ELSEIF variable BIT valeur CLEAR THEN
  {code}
ELSE
  {code}
ENDIF
```

- Valeur est comparée à variable
- Valeur peut-être une variable ou une constante
- Bit est le No de bit a tester, SET s'il doit être à 1, CLEAR pour 0

Les ?? peuvent représenter les conditions suivantes:

=	égal à
is	égal à
<>	Différent de
>=	Plus grand ou égal à
<	Plus petit que
<=	Plus petit ou égal à

Fonction:

Compare et exécute conditionnellement des portions de code.

Information :

La commande If ...Then \Elseif \Else \Endif est utilisée pour tester des états entrées-sorties (ou des variables en général), sous certaines conditions. Si ces conditions sont vérifiées, la section de code intercalée est exécutée et le pointeur saute en Endif. Dans le cas contraire, le programme teste le Endif suivant, et ainsi de suite.

La section de code suivant un Else ne sera exécutée que si aucune condition spécifiés dans le If ou les Elseif n'a été vérifiée.

Lors du test de broches d'entrée, il est nécessaire d'utiliser la dénomination pin1, pinC.2, etc ... Et non le simple N° de port. Ainsi, la syntaxe doit-être "if pinC.2=1 Then" et non "If 2=1 Then".

Notez que :

```
If b0 > 1 Then (goto) adresse  ;ligne à structure simple
If b0 > 1 Then gosub adresse   ;ligne à structure simple
If b0 > 1 Then ...else...endif ;ligne à structure multiple
```

Sont trois structures différentes, qui ne peuvent être combinées. Le code suivant :

```
if b0 > 1 then goto adresse else goto adresse2
```

est invalide, dans la mesure ou le compilateur est incapable de séparer les structures.

(c) Revolution Education Ltd.
All rights reserved

Autre exemple:
```
if b0 >1 then goto adresse : else : goto adresse2
```

Ou:
```
if b0 >1 then : goto adresse : else : goto adresse2
```

La syntaxe correcte est la suivante:
```
If b0 > 1 then
     goto adresse
else
     goto adresse2
end if
```

Ou:
```
If b0 > 1 then : goto adresse : else : goto adresse2 : endif
```

Les séparateurs ":" utilisés dans cet exemple rendent la syntaxe correcte, pour le compilateur.

(c) Revolution Education Ltd.
All rights reserved

2.54 if...then {goto}

2.55 If...and/or..then {goto}

106

Syntaxe:

IF valeur variable= ? ?{AND/OR valeur variable= ? ?} THEN goto adresse

IF valeur variable BIT

- Valeur est comparée à variable
- Valeur peut-être une variable ou une constante
- Adresse indique l'emplacement du saut a effectuer, si la condition est remplie

L'utilisation d'un goto est ici optionnelle

Les ?? peuvent représenter les conditions suivantes:

=	égal à
<>	Différent de
>=	Plus grand ou égal à
<	Plus petit que
<=	Plus petit ou égal à

Fonction:

Compare deux valeurs, et saute conditionnellement à "adresse"

Information :

La commande If ...Then est utilisée pour tester des états entrées-sorties (ou des variables en général), sous certaines conditions. Si ces conditions sont vérifiées, le pointeur saute à l'adresse spécifiée. Dans le cas contraire, le programme continue, à l'instruction suivante.

Lors du test de broches d'entrée, il est nécessaire d'utiliser la dénomination pin1, pinC.2, etc ... Et non le simple N° de port. Ainsi, la syntaxe doit-être "if pinC.2=1 Then" et non "If 2=1 Then". La commande If...Then vérifie l'état d'une entrée à un instant donné (lors de son exécution). Il est cependant conseillé de placer cette commande au sein d'une boucle, appelée régulièrement afin de vérifier constamment l'état d'un port. Pour une surveillance permanente d'une ligne d'entrée, veuillez vous reporter à l'utilisation des interruptions, décrite dans le paragraphe consacré à "Setint"

Exemple :

```
main :
 if pinC.0 = 1 then            ; teste si la broche C.0 est à l'état haut
     goto flsh                 ; si oui, saut vers flsh
 end if                        ; fin du test
 goto main                     ; sinon, retour au départ

flsh:
 high B.1                      ; met le port B.1 à l'état haut
 pause 5000                    ; attente 5 secondes
 low B.1                       ; met le port B.1 à l'état bas
 pause 5000                    ; attente 5 secondes
 goto main                     ; retour au début
```

2.56 If porta ... then {goto}

2.57 If portc... then {goto}

Syntaxe :

IF PORTA broche ? ? valeur {AND/OR variable ? ? valeur...} THEN adresse

IF PORTC broche ? ? valeur {AND/OR variable ? ? valeur...} THEN adresse

- Broche est le porta/ portc à tester
- Valeur est une variable, ou une constante
- Adresse spécifie au pointeur ou sauter, si la condition vérifiée est vraie.

L'instruction **goto** *après le* **then** *est ici optionnelle*

Les ?? peuvent représenter les conditions suivantes:

=	égal à
is	égal à
<>	Différent de
>=	Plus grand ou égal à
<	Plus petit que
<=	Plus petit ou égal à

Fonction:

Compare et saute conditionnellement à adresse

Information :

Cette commande ne peut être utilisée qu'avec les anciennes puces de type 28X/X1.Pour les puces plus récentes, utilisez la notation directe PORT.PIN. Ex: If pinC.1=1 then ...

Certaines puces PICAXE disposent de fonctions d'entrées complémentaires. Dans ce cas, les mots clés PORTA et PORTC sont insérés après le IF afin de diriger toute la ligne de commandes sur le port désiré. Il est possible d'utiliser des AND ou OR au sein de la ligne de commande, mais toutes les broches testées devront appartenir au même port, il est impossible de mélanger deux port, sur une même ligne de commande.

La commande if...then ne vérifie l'état d'une broche d'entrée que lors de son évocation. Pour une vérification régulière, il est habituel de placer cette commande au sein d'une boucle. Pour une vérification permanente, l'usage d'une interruption est détaillée, dans le paragraphe consacré à l'instruction Setint

Exemple:

Test d'un porta au sein d'une boucle.

```
Main :
 If porta pin0 = 1 then flsh          ;saute à l'adresse flsh si pin0 est à 1
 Goto main

Flsh : high1                          ;passe le port 1 à 1
 Pause 5000                           ;attente 5 secondes
 Low 1                                ;passe le port 1 à 0
 Goto main                            ;retour au départ
```

2.58 If ... then exit

2.59 If ... and/or then exit

Syntaxe :

IF variable ?? valeur {AND/OR variable ?? valeur...} THEN EXIT

IF variable BIT valeur SET/CLEAR THEN EXIT (puces X1 et X2 seulement)

- Variable est comparée à valeur
- Valeur est une variable ou une constante.

Les ?? peuvent représenter les conditions suivantes:

=	égal à
is	égal à
<>	Différent de
>=	Plus grand ou égal à
<	Plus petit que
<=	Plus petit ou égal à

Fonction:

Compare et sort conditionnellement d'une boucle do...loop ou for...next.

Information :

La commande if...then exit est utilisée pour tester l'état d'un port, ou plus généralement d'une variable, dans certaines conditions.Si ces conditions sont remplies, la boucle est prématurément close.

De multiples opérateurs peuvent être utilisés, tels que AND et OR. (voir if...then goto)

Exemple:

Test d'un port au sein d'une boucle.

```
Do
     If pinC.0=1 then exit      ;sort de la boucle si pinC.0 est à 1
loop
```


(c) Revolution Education Ltd.
All rights reserved

2.60 If ... then gosub

2.61 If ... and/or then gosub

Syntaxe :

IF variable ? ? valeur {AND/OR variable ? ? valeur...} GOSUB adresse

IF variable BIT valeur SET/CLEAR THEN GOSUB adresse (puces X1 et X2 seulement)

- Variable est comparée à valeur
- Valeur est une variable ou une constante.
- Adresse est la destination du saut, si la condition est remplie.

Les ?? peuvent représenter les conditions suivantes:

=	égal à
is	égal à
<>	Différent de
>=	Plus grand ou égal à
<	Plus petit que
<=	Plus petit ou égal à

Fonction:

Compare et exécute conditionnellement un saut vers adresse.

Information :

La commande if...then exit est utilisée pour tester l'état d'un port, ou plus généralement d'une variable, dans certaines conditions. Si ces conditions sont remplies, la sous-routine se trouvant à adresse est exécutée. Dans le cas contraire, le programme se poursuit normalement. Toute sous-procédure exécutée renvoie le pointeur à la ligne suivant le gosub.

Lors du test de broches d'entrée, il est nécessaire d'utiliser la dénomination pin1, pinC.2, etc ... Et non le simple N° de port. Ainsi, la syntaxe doit-être "if pinC.2=1 Then" et non "If 2=1 Then". La commande If...Then vérifie l'état d'une entrée à un instant donné (lors de son exécution). Il est cependant conseillé de placer cette commande au sein d'une boucle, appelée régulièrement afin de vérifier constamment l'état d'un port. "

De multiples opérateurs peuvent être utilisés, tels que AND et OR. (voir if...then goto)

Exemple:

Test d'un port au sein d'une boucle.

```
Main :
     If pinC.0=1 then gosub fslh      ;exécution de la sous-routine
                                      ;fslh si pinC.0 est à 1
     Goto main                        ;sinon, retour

Flsh : High B.1                       ;place B.1 à l'état 1
     Pause 5000                       ;attente 5 secondes
     Low B.1                          ;place B.1 à l'état 0
Return                                      ;sortie sous-routine
```

Porte ET à deux entrées

```
If pinC.1=1 and pinC.2=1 Then gosub adresse
```

Porte ET à trois entrées

```
If pinC.0=1 and pinC.1=1 and pinC.2=1 Then gosub adresse
```

Porte OU à deux entrées

```
If pinC.1=1 or pinC.2=1 Then gosub adresse
```

Fenêtre pour valeurs analogiques

```
Readadc 1,b1
If b1>= 100 and b1<= 200 then gosub adresse
```

La lecture complète d'un port peut être réalisée, en utilisant la variable 'pins'

```
     If pins = %10101010 then gosub adresse
```

Même méthode, mais en masquant certains bits

```
     Let b1= pins & %11000000
     If b1= %11000000 then gosub adresse
```

Le mot 'is' (=) on et 'is' off peut également être employé (pour les plus jeunes)

```
Loop1 :
     If pin0 is on then gosub flsh     ;appelle flsh si pin0 est à 1
     Goto loop1                        ;repart au début

Flsh :
     High B.1                          ;passe B.1 à 1
     Pause 5000                        ; attente 5 secondes
     Low B.1                           ;passe B.1 à 0
     Return                            ;retour
```


(c) Revolution Education Ltd.
All rights reserved

2.62 inc

111

Syntaxe :

INC variable

- Variable est la variable à incrémenter

Fonction:

Incrémente (ajoute 1) à la valeur de la variable

Information :

Cette commande est un racourci pour la syntaxe **let variable = variable + 1**

Exemple:

```
For b1 = 1 to 5
        Inc b2
Next b1
```


(c) Revolution Education Ltd.
All rights reserved

2.63 infrain

112

Cette commande est obsolète, merci d'utiliser irin en lieu et place.

Syntaxe :

INFRAIN

Fonction:

Attend jusqu'à réception d'une nouvelle donnée infrarouge.

Description :

Cette commande était originellement utilisée pour l'attente de nouvelles commandes infrarouges, issues d'une télécommande de téléviseur. Elle peut aussi être utilisée en réception de données émises par un autre PICAXE.

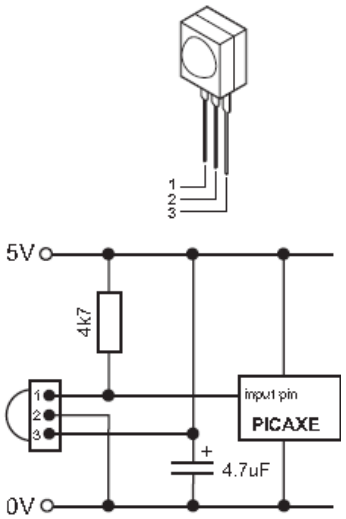
Tous les processus sont arrêtés, jusqu'à réception d'un signal valide.

La valeur du signal reçu est placée dans une variable pré-définite, 'infra'.

L'entrée infrarouge est l'entrée 0, sur toutes les puces supportant cette commande. Voir également infrain2.

La variable 'infra' est indépendante des autres variables.

En cas d'utilisation de cette commande, vous pouvez être amené a effectuer une RAZ matérielle afin de pouvoir re-télécharger un programme depuis le PC. Voir la section téléchargements série, pour plus de détails.



Effet de l'augmentation de la vitesse d'horloge :

Cette commande ne fonctionne qu'à 4 Mhz.

Utilisation de la télécommande TVR010 :

Le tableau ci-contre détaille les valeurs placées dans 'infra', selon les touches pressées.

Avant usage (ou après un changement de piles), la TVR010 doit être programmée avec les codes Sony, comme suit :

Insérez trois piles type AAA de préférence alcalines, dans la télécommande.

Appuyez sur la touche 'C', la LED doit s'allumer

Appuyez sur la touche '2', la LED doit clignoter

Appuyez sur la touche '1', la LED doit clignoter

Appuyez sur la touche '2', la LED doit clignoter, puis s'éteindre

Key	Value
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
P+	10
0	11
V+	12
P-	13
10+	14
V-	15
Mute	16
Power	17

Exemple :

```
main:
infrain                 ;attends un signal infrarouge
if infra = 1 then swon1 ;allume 1
if infra = 2 then swon2 ;allume 2
if infra = 3 then swon3 ;allume 3
if infra = 4 then swoff1 ;éteint 1
if infra = 5 then swoff2 ;éteint 2
if infra = 6 then swoff3 ;éteint 3
goto main
swon1: high 1
goto main
swon2: high 2
goto main
swon3: high 3
goto main
swoff1: low 1
goto main
swoff2: low 2
goto main
swoff3: low 3
goto main
```

(c) Revolution Education Ltd.
All rights reserved

2.64 infrain2

114

Cette commande est obsolète, merci d'utiliser irin en lieu et place.

Syntaxe :

INFRAIN2

Fonction:

Attend jusqu'à réception d'une nouvelle donnée infrarouge.

Description :

Cette commande était originellement utilisée pour l'attente de nouvelles commandes infrarouges, issues d'une télécommande de téléviseur. Elle peut aussi être utilisée en réception de données émises par un autre PICAXE.

Tous les processus sont arrêtés, jusqu'à réception d'un signal valide.

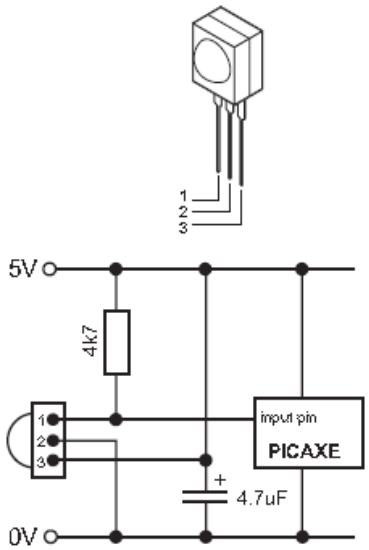
La valeur du signal reçu est placée dans une variable pré-définite, 'infra'.

Cette valeur peut-être comprise entre 0 et 127.

Sur les PICAXE 08M/14M/20M 'infra' est l'autre nom de b13-c'est la même variable. Il n'y a qu'une entrée infrarouge par puce, consultez les schémas et brochages des differents types de PICAXEs. Sur les puces de type M2,

le compilateur sort une commande irin en utilisant b13

Note du traducteur : sûrement une erreur. Irin est une commande 'entrante'.



En cas d'utilisation de cette commande, vous pouvez être amené a effectuer une RAZ matérielle afin de pouvoir re-télécharger un programme depuis le PC. Voir la section téléchargements série, pour plus de détails.

Effet de l'augmentation de la vitesse d'horloge :

Cette commande ne fonctionne qu'à 4 Mhz. Utilisez la commande 'setfreq m4' avant usage en cas d'horloge à 8 Mhz.

Exemple :

```
main:
        infrain2                    ; attente signal
        if infra = 1 then swon1     ; allume 1
        if infra = 4 then swoff1    ; éteint 1
        goto main

swon1: high 1
        goto main

swoff1: low 1
        goto main
```


(c) Revolution Education Ltd.
All rights reserved

2.65 infraout

115

Cette commande est obsolète, merci d'utiliser irin en lieu et place.

Syntaxe :

INFRAOUT matériel, donnée

- Device est une variable ou une constante désignant le No d'identification de matériel (de 1 à 31)
- Donnée est une constante ou une variable (de 0 à 127)

Fonction:

Transmission d'un signal infrarouge codé SIRC, sur porteuse à 38 Khz.

Description :

Cette commande est destinée à l'envoi de commandes infrarouges vers un matériel de type Sony. (ou à un autre PICAXE, utilisant les commandes infrain ou infrain2). Les données sont transmises via une LED infrarouge (connectée à la sortie 0), en utilisant la codification SIRC (Sony Infra Red Control)

Matériel - 5 bits (identification de 0 à 31)

Data - 7 bits (0 à 127)

Lors de l'utilisation de cette commande, en conjonction avec un autre PICAXE, l'identification matérielle doit être le 1 (équi TV). La commande infraout peut transmettre n'importe quelle valeur entre 0 et 127.. Le protocole Sony n'utilisant que 7 bits, les valeurs supérieures à 127 seront erronées.

Exemple d'une commande correcte :

```
Infraout 1,x      ;(ou x est compris entre 0 et 127)
```

Start	Data0	Data1	Data2	Data3	Data4	Data5	Data6	ID0	ID1	ID2	ID3	ID4
2.4ms	1.2 or 0.6ms	1.2 or 0.6ms	1.2 or 0.6ms	1.2 or 0.6ms	1.2 or 0.6ms	1.2 or 0.6ms	1.2 or 0.6ms	1.2 or 0.6ms	1.2 or 0.6ms	1.2 or 0.6ms	1.2 or 0.6ms	1.2 or 0.6ms

Protocole Sony SIRC :

Ce protocole utilise un signal infrarouge modulé à 38 Khz, composé d'un bit de départ (2,4ms) suivit de 12 bits (7 de données, 5 d'identification). Un niveau logique 1 est transmis par une impulsion de 1,2ms, un niveau 0 par une impulsion de 0,6 ms. Chaque bit transmit est séparé par un 'silence' de 0,6 ms.

Exemple :

```
for b1 = 1 to 10
     infraout 1,5
     pause 45
next b1
```

Interaction entre les commandes infrain, infrain2 et infraout.
Infrain et infraout

La commande d'origine infrain était destiné a être utilisée en conjonction avec les télécommandes de téléviseurs de type TVR010.
Seuls étaient donc pris en charge les 17 commandes de ce matériel,
(1-9, 0.10+, P+, P-, V+, V-, MUTE, PWR), avec des valeur de 1 à 17.

La commande peut être utilisée pour 'émuler' un TVR010. Les valeurs transmises par chaque bouton de la télécommande sont listées dans le tableau ci-contre.

TVR010 TV Remote Control	infraout / irout command	infrain variable data value	infrain2, Irin variable data value
1	infraout 1,0	1	0
2	infraout 1,1	2	1
3	infraout 1,2	3	2
4	infraout 1,3	4	3
5	infraout 1,4	5	4
6	infraout 1,5	6	5
7	infraout 1,6	7	6
8	infraout 1,7	8	7
9	infraout 1,8	9	8
P+	infraout 1,16	10	16
0	infraout 1,9	11	9
V+	infraout 1,18	12	18
P-	infraout 1,17	13	17
10+	infraout 1,12	14	12
V-	infraout 1,19	15	19
MUTE	infraout 1,20	16	20
PWR	infraout 1,21	17	21

Infrain2 et infrout :
La commande infrain2 supporte n'importe quel code TV valide entre 0 et 127.
La commande infrout peut être utilisée pour transmettre n'importe quelle valeur de 0 à 127.
(Pas au delà, voir protocole Sony, sur 7 bits)

Exemple d'une commande correcte :

```
Infraout 1,x      ;(où x est compris entre 0 et 127)
```

Effet de la modification de la vitesse d'horloge :

Ces commandes ne fonctionnent qu'à 4 Mhz.

Numéros d'identification matériels communs, chez Sony :

TV	1	VTR3	11
VTR1	2	Surround sound	12
Text	3	Audio	16
Ecran large	4	Lecteur CD	17
MDP/disque laser	6	Pro-logic	18
VTR2	7	DVD	26

Données infrarouges selon bouton pressé pour Sony :

000	1 bouton
001	2 bouton
002	3 bouton
003	4 bouton
004	5 bouton
005	6 bouton
006	7 bouton
007	8 bouton
008	9 bouton
009	10 bouton/0 bouton
011	Entrée
016	Canal supérieur
017	Canal inférieur
018	+ volume
019	- volume
020	Silence
021	Marche
022	Reset TV
023	Mode audio: Mono/SAP/Stéréo
024	Image haute
025	Image basse
026	+ Couleur
027	- couleur
030	+ lumière
031	- lumière
032	+ teinte
033	- teinte
034	+ netteté
035	- netteté
036	Choix tuner
038	Balance gauche
039	Balance droite
041	Surround M/A
042	Aux/Ant
047	Arrêt
048	Temps d'affichage
054	Temps sommeil
058	Affichage canal
059	Saut canal
064	Entrée vidéo 1
065	Entrée vidéo 2
066	Entrée vidéo 3
074	Réducteur de bruit M/A
078	Câble/hertzien
079	Filtre suppresseur M/A
088	Image dans l'image, canal supérieur
089	Image dans l'image canal inférieur
091	Image dans l'image M
092	Figeage écran
094	Image dans l'image position
095	Basculement image/image dans l'image
096	Guide
097	Paramètres vidéo
098	Paramètres audio
099	Sortie paramétrage
107	Programmation automatique
112	+ aigus
113	- aigus
114	+ basses
115	- basses
116	Touche +
117	Touche -
120	Ajout canal
121	Suppression canal
125	Trinitone M/A
127	Affiche un test rouge sur écran

Données infrarouges selon bouton pressé pour Sony (identification matériel 2 ou 7):

000	1 bouton
001	2 bouton
002	3 bouton
003	4 bouton
004	5 bouton
005	6 bouton
006	7 bouton
007	8 bouton
008	9 bouton
009	10 bouton/0 bouton
010	11 bouton
011	12 bouton
012	12 bouton
013	14 bouton
021	X 2 mode son large
022	Marche
023	Ejection
024	Canal droit/canal gauche/stéréo
025	Arrêt défilement
026	Pause
027	Démarrage
028	Retour arrière
029	FF
032	Enregistrement
035	Pause engage
040	X 1/5
041	Retour arrière vidéo
042	Avance vidéo
045	TV/VTR
047	VTR depuis TV
048	Arrêt
049	Retour arrière lent
060	Avance lente
070	Auxiliaire
078	TV/VTR
083	Index (balayage)
106	Edition
107	Marque

2.66 input

120

Syntaxe :

INPUT broche, broche, broche

- Broche est une variable ou une constante qui spécifie les broche à utiliser.

Fonction:

Positionne une broche en mode entrée.

Information :

Cette commande n'est nécessaire qu'avec les µP disposant de broches programmables en entrée/sortie. Elle peut être utilisée pour changer une broche d'entrée en broche de sortie, et vice et versa.

Lors de la première mise sous tension, toutes les broches sont configurées par défaut en entrées (sauf dans le cas de broche non programmable).

Les broches non programmables ne sont pas affectées par cette commande. Ces broches sont :

08, 08M, 08M2	0 = figée en sortie	3 = figée en entrée
14M2	B.0 = figée en sortie	C.3 = figée en entrée
18M2	C.3 = figée en sortie	C.4, C.5 = figées en entrées
20M2, 20X2	A.0 = figées en sorties	C.6 = figée en entrée
28X2, 40X2	**A.4 = figée en sortie**	

Exemple :

```
main:
    input B.1           ; force B.1 en entrée
    reverse B.1         ; inverse le sens de fonctionnement de B.1
    reverse B.1         ; inverse le sens de fonctionnement de B.1
    output B.1          ; force B.1 en sortie
```


(c) Revolution Education Ltd.
All rights reserved

2.67 inputtype

121

Syntaxe :

INPUTTYPE masque

Masque est une variable ou une constante qui spécifie le mode d'entrée d'une broche.

Fonction:

Adapte matériellement le type d'entrée souhaitée, en mode TTL ou TRIGGER DE SCHMIDT.

Information :

Les entrées tout ou rien d'un µP peuvent être de deux types, soit compatibles TTL, soit compatible TDS. Sur la plupart des puces PICAXEs, ce choix est figé lors de la fabrication (hardware), et ne peut être modifié. Beaucoup de puce intègre un mixe de ces deux technologies. Voyez les datasheet correspondant pour plus de détails concernant chaque type de PICAXE.

Cependant, grâce aux avancées de la technologie silicium, l'une des plus récente puce M2 peut voir ses entrée adaptées à chacune des deux technologies (TTL ou TDS), selon le souhait de l'utilisateur.

Masque est une valeur type word, dont les bits de 0 à 7 correspondent aux broches C.0 à C.7, et les bits 8 à 15 aux broches C.0 à C.7 En positionnant n'importe lequel (ou plusieurs) de ce (ces) bit(s) à l'état haut, l'entrée concernée passe en technologie TDS, ou en technologie TTL, en plaçant le(s) bit(s) à l'état bas.(A la mise sous tension, toutes les broches sont en TTL).

La différence technique entre les modes TTL et TDS est la suivante :

	Exemple:	5V	3V
TRIGGER DE SCHMIDT (TDS)			
Etat haut, si supérieur de 0.8 fois V alimentation		> 4V	>2,4V
Etat bas, si inférieure de 0,2 fois V alimentation		< 1V	< 0,6V
TTL (tension alim>4,5V)			
Etat haut, si supérieure à 2V		>2V	------
Etat bas, si inférieure à 0,8V		< 0,8V	------
TTL (tension alim < 4,5V)			
Etat haut, si supérieure à 0,25 fois V alim + 0,8V		--------	> 1,55V
Etat bas, si inférieure à 0,15 fois V alim		--------	< 0,45V

Des valeurs comprises entre ces maxi-mini sont considérées comme "flottantes" et ne peuvent être prise en compte de façon fiable.

De manière générale, les entrées en mode TTL sont considérées comme plus versatiles, et garantissent un basculement franc dès 2 V, pour une tension d'alimentation de 5V.

Note du traducteur: Plutôt 2,5 V, pour une garantie de basculement.

Cependant, le mode TDS peut parfois s'avérer utile.

Exemple :

```
main:
inputtype %0000000000001111 ; passe broches B.0 à B.3 mode TDS
inputtype %0000111100000000 ; passe broches C.0 à C.3 mode TDS
```


(c) Revolution Education Ltd.
All rights reserved

Input Pin Types:

08M2 08M 08
Serin TTL TTL TTL
C.1 TTL TTL TTL
C.2 ST ST ST
C.3 TTL TTL TTL
C.4 TTL TTL TTL

14M2* 14M
Serin TTL TTL
B.0 TTL n/a
B.1 TTL n/a
B.2 TTL n/a
B.3 TTL n/a
B.4 TTL n/a
B.5 TTL n/a
C.0 TTL TTL
C.1 TTL TTL
C.2 TTL TTL
C.3 TTL TTL
C.4 TTL TTL

* les broches du 14M2 peuvent être reconfigurées via 'inputtype'

18M2 18X 18M 18A 18
Serin TTL ST ST ST ST
B.0 TTL n/a n/a n/a n/a
B.1 TTL n/a n/a n/a n/a
B.2 TTL n/a n/a n/a n/a
B.3 TTL n/a n/a n/a n/a
B.4 TTL n/a n/a n/a n/a
B.5 TTL n/a n/a n/a n/a
B.6 TTL n/a n/a n/a n/a
B.7 TTL n/a n/a n/a n/a
C.0 TTL TTL TTL TTL ST
C.1 TTL TTL TTL TTL ST
C.2 TTL TTL TTL TTL ST
C.5 TTL n/a n/a n/a n/a
C.6 TTL ST ST ST ST
C.7 TTL ST ST ST ST

20X2 20M2* 20M
Serin TTL TTL TTL
B.0 TTL TTL n/a
B.1 TTL TTL n/a
B.2 ST TTL n/a
B.3 ST TTL n/a
B.4 ST TTL n/a
B.5 TTL TTL n/a
B.6 TTL TTL n/a
B.7 TTL TTL n/a
C.0 TTL TTL TTL
C.1 ST TTL ST
C.2 ST TTL ST
C.3 ST TTL ST
C.4 ST TTL ST
C.5 ST TTL ST
C.6 TTL TTL TTL
C.7 TTL TTL TTL

* les broches du 20M2 peuvent être reconfigurées via 'inputtype'

28X2 28X2-5V 28X2-3V 28X1 28X 28A 28
Serin ST ST ST ST ST ST ST
A.0 TTL TTL TTL TTL TTL ADC ADC
A.1 TTL TTL TTL TTL TTL ADC ADC
A.2 TTL TTL TTL TTL TTL ADC ADC
A.3 TTL TTL TTL TTL TTL ADC ADC

(c) Revolution Education Ltd.
All rights reserved

B.0 TTL TTL TTL n/a n/a n/a n/a
B.1 TTL TTL TTL n/a n/a n/a n/a
B.2 TTL TTL TTL n/a n/a n/a n/a
B.3 TTL TTL TTL n/a n/a n/a n/a
B.4 TTL TTL TTL n/a n/a n/a n/a
B.5 TTL TTL TTL n/a n/a n/a n/a
B.6 TTL TTL TTL n/a n/a n/a n/a
B.7 TTL TTL TTL n/a n/a n/a n/a
C.0 TTL ST ST ST ST ST ST
C.1 TTL ST ST ST ST ST ST
C.2 TTL ST ST ST ST ST ST
C.3 TTL ST ST ST ST ST ST
C.4 TTL ST ST ST ST ST ST
C.5 TTL ST ST ST ST ST ST
C.6 TTL ST ST ST ST ST ST
C.7 TTL ST ST ST ST ST ST

40X2 40X2-5V 40X2-3V 40X1 40X
Serin ST ST ST ST ST
A.0 TTL TTL TTL TTL TTL
A.1 TTL TTL TTL TTL TTL
A.2 TTL TTL TTL TTL TTL
A.3 TTL TTL TTL TTL TTL
A.5 TTL ST ST ADC ADC
A.6 TTL ST ST ADC ADC
A.7 TTL ST ST ADC ADC
B.0 TTL TTL TTL n/a n/a
B.1 TTL TTL TTL n/a n/a
B.2 TTL TTL TTL n/a n/a
B.3 TTL TTL TTL n/a n/a
B.4 TTL TTL TTL n/a n/a
B.5 TTL TTL TTL n/a n/a
B.6 TTL TTL TTL n/a n/a
B.7 TTL TTL TTL n/a n/a
C.0 TTL ST ST ST ST
C.1 TTL ST ST ST ST
C.2 TTL ST ST ST ST
C.3 TTL ST ST ST ST
C.4 TTL ST ST ST ST
C.5 TTL ST ST ST ST
C.6 TTL ST ST ST ST
C.7 TTL ST ST ST ST
D.0 TTL TTL TTL TTL TTL
D.1 TTL TTL TTL TTL TTL
D.2 TTL TTL TTL TTL TTL
D.3 TTL TTL TTL TTL TTL
D.4 TTL TTL TTL TTL TTL
D.5 TTL TTL TTL TTL TTL
D.6 TTL TTL TTL TTL TTL
D.7 TTL TTL TTL TTL TTL

(c) Revolution Education Ltd.
All rights reserved

2.68 irin

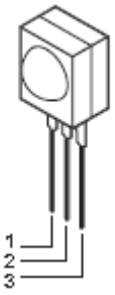
125

Syntaxe :

IRIN broche, variable

IRIN [dépassement], broche, variable

IRIN [dépassement, adresse], broche, variable

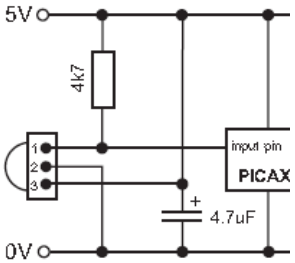


Dépassement est une variable ou une constante qui spécifie le temps maximum d'attente en ms.

Adresse est un label, qui spécifie ou sauter, lors d'un dépassement du temps fixé par dépassement.

Broche est une variable, ou une constante, qui spécifie quelle broche utiliser.

Variable reçoit la donnée entrante.



Fonction:

Attente de la réception d'une nouvelle commande infrarouge .

Cette commande est similaire à infrain2, mais peut être utilisée sur toute broche d'entrée.

Description :

Cette commande est utilisée pour l'attente d'un signal infrarouge émanant d'un autre PICAXE. Elle peut également être utilisée, pour recevoir des signaux IR, depuis une télécommande de téléviseur (en remplacement de infrain).

Tous les processus stoppent, en attendant la réception du signal, mais, après un délai fixé par dépassement, le programme saute au label Définit par adresse.

La valeur reçue est placée dans variable. Elle doit être comprise entre 0 et 127.

Voir la commande infraout, pour plus de détails, sur la télécommande TVR010.

Start	Data0	Data1	Data2	Data3	Data4	Data5	Data6	ID0	ID1	ID2	ID3	ID4
2.4ms	1.2 or 0.6ms	1.2 or 0.6ms	1.2 or 0.6ms	1.2 or 0.6ms	1.2 or 0.6ms	1.2 or 0.6ms	1.2 or 0.6ms	1.2 or 0.6ms	1.2 or 0.6ms	1.2 or 0.6ms	1.2 or 0.6ms	1.2 or 0.6ms

Afin de remplacer une commande infrain ou infrain2, par une commande irin, utilisez ces deux lignes :

```
Symbol infra = b13              ; définit la variable infra
Irin C.0, infra                 ; lecture C.0 et stockage en infra
```

Effet de l'augmentation de la vitesse d'horloge :

Cette commande utilisera automatiquement le résonateur interne à 4 Mhz, pour fonctionner correctement.

Exemple :

```
main:
        irin [1000,main],C.3,b0         ;attente signal
        if b0 = 1 then swon1            ;B.1 à 1
        if b0 = 4 then swoff1           ;B.1 à 0
        goto main
        swon1: high B.1
        goto main
        swoff1: low B.1
        goto main
```

TVR010 TV Remote Control	irout command	infrain variable data value	infrain2, irin variable data value
1	irout pin,1,0	1	0
2	irout pin,1,1	2	1
3	irout pin,1,2	3	2
4	irout pin,1,3	4	3
5	irout pin,1,4	5	4
6	irout pin,1,5	6	5
7	irout pin,1,6	7	6
8	irout pin,1,7	8	7
9	irout pin,1,8	9	8
P+	irout pin,1,16	10	16
0	irout pin,1,9	11	9
V+	irout pin,1,18	12	18
P-	irout pin,1,17	13	17
10+	irout pin,1,12	14	12
V-	irout pin,1,19	15	19
MUTE	irout pin,1,20	16	20
PWR	irout pin,1,21	17	21

(c) Revolution Education Ltd.
All rights reserved

2.69 irout

127

Syntaxe :

IROUT broche, matériel, donnée

Broche est une variable, ou une constante, qui spécifie quelle broche utiliser.

Matériel est une variable, ou une constante, qui spécifie le No du matériel.

Donnée est une variable, ou une constante (de 0 à 127)

Fonction:

Transmet un signal infrarouge modulé à 38 Khz

Cette commande est similaire à infraout, mais peut être utilisée sur toute broche de sortie.

Description :

Cette commande est utilisée pour transmettre un signal infrarouge à un appareillage Sony, (ou à un autre PICAXE utilisant les commandes irin, infrain ou infrain2). Le signal est transmis via une LED infrarouge, en utilisant le protocole SIRC (Sony Infrared Control)

Matériel - 5 bits, identification de 0 à 31

Donnée - 7 bits, valeur de 0 à 127

Dans le cas de transmission de PICAXE à PICAXE, le No d'identification doit être 1

La commande irout peut transmettre n'importe quelle valeur, comprise entre 0 et 127.

La bonne syntaxe, pour une utilisation avec les commandes infrain2/infrain/irin est la suivante :

```
Irout, 1 ,1 , x                    ; avec x compris entre 0 et 127
```

Protocol Sony SIRC:

Ce protocole utilise un signal infrarouge modulé à 38 Khz, composé d'un bit de départ (2,4ms) suivit de 12 bits (7 de données, 5 d'identification). Un niveau logique 1 est transmis par une impulsion de 1,2ms, un niveau 0 par une impulsion de 0,6 ms. Chaque bit transmit est séparé par un 'silence' de 0,6 ms.

Effet de l'augmentation de la vitesse d'horloge :

Cette commande utilisera automatiquement le résonateur interne à 4 Mhz, pour fonctionner correctement.

Exemple :

Toutes les télécommande TV du commerce répètent leur signal toutes les 45 ms, tant que le bouton est pressé. L'utilisation de PICAXE augmente la fiabilité de la transmission, et ne nécessite pas plus d'une dizaine de répétitions, pour un usage fiable.

```
for b1 = 1 to 10
        irout 1,1,5
        pause 45
next b1
```

Syntaxe :

IROUT broche, matériel, donnée

Broche est une variable, ou une constante, qui spécifie quelle broche utiliser.

Matériel est une variable, ou une constante, qui spécifie le No du matériel.

Donnée est une variable, ou une constante (de 0 à 127)

Fonction:

Transmet un signal infrarouge modulé à 38 Khz

Cette commande est similaire à infraout, mais peut être utilisée sur toute broche de sortie.

Description :

Cette commande est utilisée pour transmettre un signal infrarouge à un appareillage Sony, (ou à un autre PICAXE utilisant les commandes irin, infrain ou infrain2). Le signal est transmit via une LED infrarouge, en utilisant le protocole SIRC (Sony Infrared Control)

Matériel - 5 bits, identification de 0 à 31

Donnée - 7 bits, valeur de 0 à 127

Dans le cas de transmission de PICAXE à PICAXE, le No d'identification doit être 1

La commande irout peut transmettre n'importe quelle valeur, comprise entre 0 et 127.

La bonne syntaxe, pour une utilisation avec les commandes infrain2/infrain/irin est la suivante :

```
Irout, 1 ,1 , x                    ; avec x compris entre 0 et 127
```

Protocol Sony SIRC:

Ce protocole utilise un signal infrarouge modulé à 38 Khz, composé d'un bit de départ (2,4ms) suivit de 12 bits (7 de données, 5 d'identification). Un niveau logique 1 est transmis par une impulsion de 1,2ms, un niveau 0 par une impulsion de 0,6 ms. Chaque bit transmit est séparé par un 'silence' de 0,6 ms.

Effet de l'augmentation de la vitesse d'horloge :

Cette commande utilisera automatiquement le résonateur interne à 4 Mhz, pour fonctionner correctement.

Exemple :

Toutes les télécommandes TV du commerce répètent leur signal toutes les 45 ms, tant que le bouton est pressé. L'utilisation de PICAXE augmente la fiabilité de la transmission, et ne nécessite pas plus d'une dizaine de répétitions, pour un usage fiable.

```
for b1 = 1 to 10
        irout 1,1,5
        pause 45
next b1
```

TVR010 TV Remote Control	irout command	infrain variable data value	infrain2, irin variable data value
1	irout pin,1,0	1	0
2	irout pin,1,1	2	1
3	irout pin,1,2	3	2
4	irout pin,1,3	4	3
5	irout pin,1,4	5	4
6	irout pin,1,5	6	5
7	irout pin,1,6	7	6
8	irout pin,1,7	8	7
9	irout pin,1,8	9	8
P+	irout pin,1,16	10	16
0	irout pin,1,9	11	9
V+	irout pin,1,18	12	18
P-	irout pin,1,17	13	17
10+	irout pin,1,12	14	12
V-	irout pin,1,19	15	19
MUTE	irout pin,1,20	16	20
PWR	irout pin,1,21	17	21

(c) Revolution Education Ltd.
All rights reserved

2.70 kbin

129

Syntaxe :

KBIN variable

KBIN [dépassement], variable

KBIN [dépassement, adresse], variable

KBIN #variable (puces M2 uniquement)

KBIN [dépassement], #variable (puces M2 uniquement)

KBIN [dépassement, adresse], #variable (puces M2 uniquement)

- Variable reçoit le code touche
- Dépassement est une variable, ou une constante, qui spécifie la durée d'attente maximum en ms
- Adresse est un label, qui spécifie où sauter en cas de dépassement

Fonction:

Mise en attente, jusqu'à réception d'un nouveau code clavier. Cette commande est similaire à l'ancienne commande keyin, mais intègre l'option dépassement.

Information :

Cette commande est utilisée pour l'attente de la réception d'un code touche, depuis un clavier PC,(directement connecté au PICAXE - et non via la procédure de programmation, voir keyled, pour les détails de connexion). Tous les processus sont stoppés, jusqu'à réception d'un code clavier, si le délai dépassement est atteint, le programme saute alors à adresse.Le code clavier est stocké dans variable.

Remarque : La conception des claviers n'ayant rien de 'logique', les codes touches renvoyés n'ont rien de 'logique'. Chaque touche sera identifiée selon le tableau de la page suivante. Certaines touches renvoient deux codes, le premier ($E0) est ici ignoré par le PICAXE, seul le second est pris en compte. Tous les codes sont en hexadécimal, et doivent être précédés du signe '$' dans le code. Les touches PAUSE et PRTSCR ne peuvent être exploitées, compte tenu du nombre élevé de valeurs renvoyées. Un certain nombre d'autres touches ne fonctionneront pas, si le verrou VERNUM est activé –LED allumée- par la commande keyled.

Afin de contourner cet état de fait, la variable #variable a été ajouté à la commande d'origine, pour les puces type M2. Dans ce cas, c'est le code ASCII de la lettre de la touche pressée qui est renvoyé. Les caractères non supportés, tel que 'CRTL' renvoient le caractère ' ?' .

Pour les puces plus anciennes, vous trouverez, dans le répertoire 'exemple', comment convertir les codes caractères, grâce au balayage d'une table de correspondance, via la commande lookup (« keyin.bas »).

Effet de l'augmentation de la vitesse d'horloge :

Cette commande utilisera automatiquement le résonateur interne à 4 Mhz, pour fonctionner correctement.

Exemple:

```
main:
        kbin [1000,main],b1
        if b1 = $45 then
                low b.1
        end if
        if b1= $25 then
                high b.1
        end if
        goto main
```


(c) Revolution Education Ltd.
All rights reserved

2.71 keyin

131

Cette commande est obsolète. Merci d'utiliser kbin

Syntaxe :

KEYIN

Fonction:

Mise en attente, jusqu'à réception d'un nouveau code clavier.

Information :

Cette commande est utilisée pour l'attente de la réception d'un code touche, depuis un clavier PC, (directement connecté au PICAXE - et non via la procédure de programmation, voir keyled, pour les détails de connexion). Tous les processus sont stoppés, jusqu'à réception d'un code clavier, si le délai dépassement est atteint, le programme saute alors à adresse. Le code clavier est stocké dans la variable prédéfinie 'keyvalue'.

Remarque : La conception des claviers n'ayant rien de 'logique', les codes touches renvoyés n'ont rien de 'logique'. Chaque touche sera identifiée selon le tableau de la page suivante. Certaines touches renvoient deux codes, le premier ($E0) est ici ignoré par le PICAXE, seul le second est pris en compte. Tous les code sont en hexadécimal, et doivent être précédés du signe '$' dans le code. Les touches PAUSE et PRTSCR ne peuvent être exploitées, compte tenu du nombre élevé de valeurs renvoyées. Un certain nombre d'autres touches ne fonctionneront pas, si le verrou VERNUM est activé –LED allumée- par la commande keyled.

Après l'utilisation de cette commande, un reset matériel sera nécessaire, pour relancer un téléchargement de programme. Voir la section 'téléchargement série' pour plus de détails.

Effet de l'augmentation de la vitesse d'horloge :

Cette commande utilisera automatiquement le résonateur interne à 4 Mhz, pour fonctionner correctement.

Exemple:

```
main:
        keyin                           ;attente touche pressée
        if keyvalue = $45 then swon1    ;passe la sortie 1 à 1
        if keyvalue = $25 then swoff1   ;passe la sortie 1 à 0
        goto main

swon1: high 1
        goto main

swoff1: low 1
        goto main
```

KEY	CODE	KEY	CODE	KEY	CODE
A	1C	9	46	[	54
B	32	'	0E	INSERT	E0,70
C	21	-	4E	HOME	E0,6C
D	23	=	55	PG UP	E0,7D
E	24	\	5D	DELETE	E0,71
F	2B	BKSP	66	END	E0,69
G	34	SPACE	29	PG DN	E0,7A
H	33	TAB	0D	U ARROW	E0,75
I	43	CAPS	58	L ARROW	E0,6B
J	3B	L SHIFT	12	D ARROW	E0,72
K	42	L CTRL	14	R ARROW	E0,74
L	4B	L GUI	E0,1F	NUM	77
M	3A	L ALT	11	KP /	E0,4A
N	31	R SHFT	59	KP *	7C
O	44	R CTRL	E0,14	KP -	7B
P	4D	R GUI	E0,27	KP +	79
Q	15	R ALT	E0,11	KP EN	E0,5A
R	2D	APPS	E0,2F	KP .	71
S	1B	ENTER	5A	KP 0	70
T	2C	ESC	76	KP 1	69
U	3C	F1	05	KP 2	72
V	2A	F2	06	KP 3	7A
W	1D	F3	04	KP 4	6B
X	22	F4	06	KP 5	73
Y	35	F5	03	KP 6	74
Z	1A	F6	0B	KP 7	6C
0	45	F7	83	KP 8	75
1	16	F8	0A	KP 9	7D
2	1E	F9	01	]	5B
3	26	F10	09	;	4C
4	25	F11	78	'	52
5	2E	F12	07	,	41
6	36	PRNT SCR	??	.	49
7	3D	SCROLL	7E	/	4A
8	3E	PAUSE	??		

2.72 kbled (keyled)

133

Syntaxe :

KBLED masque

- Masque est une variable, ou une constante, qui spécifie quelle(s) LED(s) utiliser.

Fonction:

Allume ou éteint les LED clavier

Information :

Cette commande est utilisée pour contrôler l'état des LEDs sur un clavier PC, (directement connecté au PICAXE - et non via la procédure de programmation). La valeur masque détermine l'état des LEDs.

Masque doit être utilisé comme suit :

Bit 0 -Blocage défilement (1=on, 0=off)

Bit 1 -Blocage chiffres (1=on, 0=off)

Bit 2 -VerrouilMaj (1=on, 0=off)

Bits 3-6 - Inutilisés

Bit 7 - Flash (1=pas de flash, 0=flash)

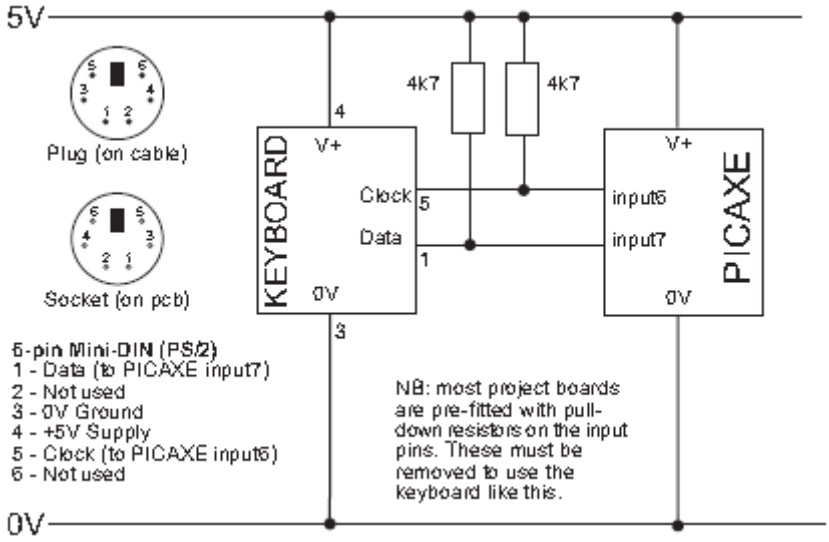
Lors d'une réinitialisation, le masque est mis à 0, donc les trois LEDs flasheront dès l'appui sur une touche. Ceci informe l'utilisateur d'une connexion correcte PICAXE-clavier. Vous pouvez désactiver cette fonction, en positionnant le bit 7 du masque à 1. Dans ce cas, les trois LED pourront être contrôlées via les bits 2 - 0 du masque.

Effet de l'augmentation de la vitesse d'horloge :

Cette commande ne fonctionne correctement qu'à une vitesse d'horloge de 4 Mhz.

Exemple:

```
main:
        keyled %10000111        ; toutes les LEDs allumées
        pause 500               ; pause 500ms
        keyled %10000000        ; toutes LEDs éteintes
        pause 500               ; pause 500 ms
        goto main               ; boucle
```



(c) Revolution Education Ltd.
All rights reserved

2.73 let

134

(tous les PICAXEs)

Syntaxe:

{LET} variable = {-} valeur?? valeur ...

- Variable : celle qui sera affectée.

- Valeur(s) sont des variables ou constantes qui affecteront la variable.

Le mot clé 'LET' est optionnel.

Fonction:

Manipule des variables .

Les opérations mathématiques sont effectuées strictement de gauche à droite. (pas de priorité)

Information:

Le microcontrôleur accepte les calculs sur des valeurs sur 16 bits (word - mot). Les entiers acceptés sont entre 0 et 65535.

Tous les calculs mathématiques peuvent être réalisés sur des valeurs byte (8 bits) donc de 0 à 255 .

Le microcontrôleur ne gère pas les fractions, ni les nombres négatifs.

Toutefois, il est parfois possible d'écrire l'équation en utilisant des entiers au lieu des fractions :

let w1 = w2 / 5.7 n'est pas valide , mais

let w1 = w2 * 10 / 57 est mathématiquement équivalent et autorisé.

Les fonctions mathématiques supportées par tous les PICAXEs sont:

```
+                   ; addition
-                   ; soustraction
*                   ; multiplication (renvoie le mot de poids faible du résultat)
**                  ; multiplication (renvoie le mot de poids fort du résultat)
/                   ; division (renvoie le quotient)
// (ou %)                   ; division modulo (renvoie le reste)
MAX                 ; limite la valeur à un maximum
MIN                 ; limite la valeur à un minimum
AND (ou &)          ; ET logique bit à bit
OR (ou | )          ; OU logique (touches Alt gr et 6 du haut du clavier AZERTY )
XOR ( ou ^)         ; XOR OU exclusif logique
NAND                ; comme son nom l'indique .. NON ET
NOR                 ; NON OU
ANDNOT (ou &/)              ; ETNON (NB ce qui est diffèrent de NAND)
ORNOT (ou |/ )      ; OUNON (NB ce qui est diffèrent de NOR)
XNOR (ou ^/)        ; NON OU Exclusif
```

Les X1 and X2 supportent également

```
<<      ; décalage à gauche
```

>> ; décalage à droite

*/ ; multiplication (renvoie le mot central d'une multiplication de deux mots de 16 bits) Voir Section 2

Les X1 et X2 supportent ces commandes :

SIN ; sinus d'un angle (0 à 65535) en degrés (est renvoyée la valeur du sinus * 100)

COS ; cosinus d'un angle en degrés (est renvoyée la valeur du cosinus * 100)

SQR ; racine carrée

INV ; inverse

NCD ; encodeur (2 à la puissance n)

DCD ; décodeur (2 à la puissance n)

BINTOBCD ; convertit une valeur binaire en BCD (binaire codé décimal)

BCDTOBIN ; convertit une valeur BCD (binaire codé décimal) en binaire

REV ; Inverse un nombre de bit d'un byte ou d'un mot : w4 = w1 REV 8 inverse les 8 bits de poids faible de w1

(pas expliqué dans la doc Rev-Ed)

DIG ; renvoie la valeur décimale d'un digit

Il n'est pas (encore) possible d'utiliser les parenthèses

w1 = w2 / (b5 + 2) n'est pas correct.

Il est nécessaire de mettre cette équation sous une forme équivalente :

w1 = b5 + 2

w1 = w2 / w1

Pour plus d'informations, voir la section Variables mathématiques de ce manuel.

Exemple:

```
main:
        inc b0                          ; incrémente  b0   (b0 = b0+1)
        B.7,(b0,50)                     ; produit un son
        if b0 > 50 then rest            ; si b0 est supérieur à 50 aller à rest
        goto main                       ;  retour au programme main

rest:
        b0 = b0 max 10                  ; limite  la valeur  de b0 à 10
        goto main                       ; retour au programme main
```

2.74 let dirs / dirsc =

136

(PICAXEs 08 – 08M – 14M – 28X – 28X1 – 40X – 40X1)

Pour les M2 et X2 voir page suivante.

Le mot clé LET étant optionnel on ne l'utilise plus dans la suite de ce texte. (NdT)

Syntaxe :

dirs = valeur

dirsc = valeur

- Valeur(s) sont des variables/constantes qui configurent le registre de direction des entrées sorties.

Fonction:

Configure les broches en tant que entrées ou sorties (dirs =) (sur 08/08M/08M2)

Configure les broches en tant que entrées ou sorties sur le port C (let dirsc =) (14M)

Configure les broches en tant que entrées ou sorties sur le port C (let dirsc =) (28X/40X)

Configure les broches en tant que entrées ou sorties sur le port C (let dirsc =) (28X1/40X1)

Information:

Certains microcontrôleurs permettent de configurer les entrées et sorties

Dans ce cas il est nécessaire de préciser au microcontrôleur quelles broches sont des entrées, et quelles sont celles qui seront des sorties. Sachant qu'à la mise sous tension, toutes les broches sont configurées en entrées.

Il y a plusieurs façons de réaliser cela :

1) Utiliser la commande reverse.

2) Utiliser une commande de sortie (high, pulsout etc) qui configurent automatiquement la broche en sortie.

3) Utiliser la commande dirs = état.

Avec cette dernière, utiliser la façon conventionnelle d'indiquer l'état en binaire.

pins 7 correspond au bit de gauche et pins 0 au bit de droite.

Un bit à 0 indique que la broche sera considérée comme une entrée. Et bien sûr le bit sera mis à 1 pour une sortie,

Notez que les PICAXEs 08 ont certaines broches préconfigurées, et non modifiables:

Pin0 est une sortie, et pin3 est une entrée.

Exemple:

```
dirs = %00000011 ; pins 0 and 1 seront des sorties
pins = %00000011 ; mets à l'état haut (1 logique) les pins 0 et 1
```

2.75 let dirsA / dirsB / dirsC / dirsD =

137

(tous les PICAXEs M2 et X2)

Syntaxe:

dirsA = valeur

dirsB = valeur

dirsC = valeur

dirsD = valeur

- valeur(s) sont des variables/constantes qui configurent le registre de direction du port.

Fonction:

Configure pins en entrées ou sorties.

Information:

Certains microcontrôleurs permettent de configurer les entrées et sorties

Dans ce cas il est nécessaire de préciser au microcontrôleur quelles broches sont des entrées, et quelles sont celle qui seront des sorties. Sachant qu'à la mise sous tension, toutes les broches sont configurées en entrées.

Il y a plusieurs façons de réaliser cela :

1) Utiliser la commande reverse.

2) Utiliser une commande de sortie (high, pulsout etc) qui configurent automatiquement la broche en sortie.

3) Utiliser la commande dirs = état.

Avec cette dernière, utiliser la façon conventionnelle d'indiquer l'état en binaire.

La broche 7 correspond au bit de gauche et la broche 0 au bit de droite.

Un bit à 0 indique que la broche sera considérée comme une entrée. Et bien sûr le bit sera mis à 1 pour une sortie.

Noter que certains PICAXEs ont certaines broches préconfigurées, et non modifiables:

Exemple:

```
dirs = %00000011 ; pins 0 and 1 seront des sorties
pins = %00000011 ; mets à l'état haut (1 logique) les pins 0 et 1
```


(c) Revolution Education Ltd.
All rights reserved

2.76 let pins / pinsc =

138

Pour les M2 et X2, voir page suivante

Syntaxe

pins = valeur

pinsc = valeur

- Valeur(s) sont des variables/constantes qui configurent le registre d'état des sorties.

Fonction:

Active ou désactive toutes les sortie sur le port principal (let pins =).

Active ou désactive toutes les sortie sur le port principal C (let pinsc =)

Information:

Les commandes HIGH et LOW peuvent être utilisées pour activer les sorties individuellement.

Toutefois, lorsqu'on travaille avec des sorties multiples, il est plus pratique de changer toutes les sorties simultanément.

De la même façon que pour DIRS, il est conseillé d'écrire la valeur d'état en binaire.

La broche 7 correspond au bit de gauche et la broche 0 au bit de droite.

Un bit à 0 indique que la broche sera mise à l'état bas (low) 0. Et bien sûr le bit sera mis à 1 pour une sortie à l'état haut (high) 1.

Ne pas confondre les ports d'entrées et les ports de sortie. Ils sont séparés sur tous les PICAXE sauf les 08.

La commande **pins = pins** recopie l'état du port d'entrée sur le port de sortie.

Il faut noter que sur les composants ayant des pins bidirectionnels, (08, 08M) cette commande ne fonctionne que sur les pins configurés en sortie.

Dans ce cas, il est nécessaire de configurer les pins en sorties. (utiliser la commande let dirs = command)

Exemple:

```
pins = %10000011        ; met les sorties 7.1.et 0 à l'état haut (1)
pause 1000              ; attends  1 seconde
pins = %00000000        ; mets toutes les sorties à l'état bas (0)
```

2.77 let pinsA / pinsB / pinsC / pinsD =

139

Syntaxe:

pinsA = valeur

pinsB = valeur

pinsC = valeur

pinsD = valeur

- Valeur (s) sont des variables / constants are variables/constants which operate on the output port.

Fonction:

Met à l'état haut ou bas toutes les sorties du port sélectionné.

Les commandes **High** et **low** peuvent être utilisée pour mettre les sorties individuelles à l'état souhaité.

Toutefois, lorsqu'on travaille sur plusieurs sorties, il est pratique de tout changer à la fois.

Utiliser pour cela la notation binaire.

La broche 7 correspond au bit de gauche et la broche 0 au bit de droite.

Si le bit est à 0, la sortie sera à un niveau bas (0) et inversement à l'état haut pour un bit à 1.

Il faut noter que cette commande ne fonctionne que sur les pins configurés en sortie.

Dans ce cas, il est nécessaire de configurer les pins en sorties. (utiliser la commande let dirsX = command)

Exemple:

```
dirsB = %10000011       ; 7,1,0 sont configurés en sorties
pinsB = %10000011       ; les pins 7,1,10 sont mis à l'état haut (1)
pause 1000              ; attente  1 seconde
pinsB = %00000000       ; mets toutes les sorties à l'état bas (0)
```


(c) Revolution Education Ltd.
All rights reserved

2.78 lookdown

140

(tous les PICAXEs)

Syntaxe:

LOOKDOWN cible,(valeur0, valeur 1... valeur N),variable

- Cible est une variable/constant qui est comparée à valeur x.
- Valeurs sont des variables ou constantes.
- Variable reçoit le résultat (s'il y en a un)

Fonction:

Affecte à la variable, la position de la valeur correspondant à la cible (si une correspondance est trouvée)

Information:

La commande lookdown doit être utilisée lorsque vous avez une valeur précise à comparer à une liste de valeurs connues.
La variable cible est comparée aux valeurs entre parenthèses. Si elle correspond à la 5ième le nombre 4 est affecté à la variable.
Noter que les valeurs sont numérotées à partir de 0 et non de 1.
S'il n'y a pas de correspondance, la valeur de la variable reste inchangée.
Dans l'exemple suivant, la variable b2 contiendra la valeur 3 si b1 contient "d" et 4 si b1 contient "e".

Exemple:

```
lookdown b1,("abcde"),b2
```


(c) Revolution Education Ltd.
All rights reserved

2.79 lookup

141

(tous les PICAXEs)

Syntaxe:

LOOKUP offset,(data0,data1...dataN),variable

- Offset est une variable/constante qui spécifie quelle donnée (data0 à dataN) est affectée à variable.
- Data sont des variables ou constantes.
- Variable reçoit le résultat s'il y en a un.

Fonction:

La donnée recherchée est définie par sa position (offset) et affectée à variable (si offset est dans l'intervalle).

Description:

La commande lookup est utilisée pour affecter à une variable différentes valeurs selon leurs positions dans la table de lookup
Dans cet exemple, si b0=0 alors b1 sera égal à "a", si b0 =1 alors b1 sera égal à "b" etc.
Si offset est plus grand que le nombre d'entrées de la table, la valeur de variable reste inchangée.
Chaque lookup est limitée à 256 entrées, mais chaque entrée peut être un bit, un byte, un mot (word) une constante ou une variable).

Exemple:

```
main:
        lookup b0,("abcde"),b1      ; Place dans la variable b1 le caractère ASCII de position b0
        inc b0                      ; incremente b0
        if b0 < 4 then main         ; boucle
        end
```


(c) Revolution Education Ltd.
All rights reserved

2.80 low

142

(tous les PICAXEs)

Syntaxe:

LOW pin {,pin,pin...}

- Pin is a variable/constant which specifies the i/o pin to use.

Fonction:

Positionne une broche en sortie et la met à l'état bas (0).

Information:

La commande Low met une sortie à l'état bas (0)
Sur les microcontrôleurs avec les entrées / sorties configurables cette commande configure automatiquement la broche en sortie.

Exemple:

```
main:
high B.1              ; met B.1 à l'état haut
pause 5000            ; attend 5 secondes
low B.1               ; met B.1 à l'état bas
pause 5000            ; attend 5 secondes
goto main             ; retour au point de départ
```

2.81 low portc

143

(PICAXEs 14M, 28X, 28X1, 40X, 40X1)

Syntax:

LOW PORTC pin {,pin,pin...}

- Pin est une variable/constante désignant l'entrée/sortie à utiliser.

Fonction:

Met une broche du port C à l'état bas.
Cette commande n'est utilisée que sur les anciens 14M et les X et X1.
Pour les nouveaux M2 and X2 utiliser la notation PORT.PIN par exemple **low C.2**

Information:

La commande high positionne une sortie du portc à l'état haut.

Exemple:

```
main:
high portc 1          'met la sortie 1 à l'état haut
pause 5000            'attend 5 secondes
low portc 1           'met la sortie 1 à l'état bas
pause 5000            'attend 5 secondes
goto main             'retour au point de départ
```


(c) Revolution Education Ltd.
All rights reserved

2.82 nap

143

(tous les PICAXEs)

Syntaxe:

NAP period

- Period est une variable / constant qui définit la durée de mise en consommation réduite (normalement 0-7, mais les PICAXE M2 acceptent de 0 à 14.)

Period	Time Delay
0	18ms
1	32ms
2	72ms
3	144ms
4	288ms
5	576ms
6	1.1s
7	2.3s
8	4s
9	8s
10	16s
11	32s
12	64s (1 min)
13	128s (2 mins)
14	256s (4 mins)

Function:

Met en sommeil sur une période courte. La consommation d'énergie est réduite mais la précision sur certaines durées est perdue. Un plus long délai est possible avec la commande **sleep**.

Information:

Quand en mode d'économie d'énergie, tous les timers sont inactifs, entraînant l'arrêt des commandes pwm et servo (voir la commande 'doze'). La durée approximative est donnée dans le tableau ci contre.

Compte tenu des tolérances des timers internes, ces temps sont à 50 à 100% près. La température extérieure affecte ces tolérances. Aussi, ne pas utiliser cette commande dans un projet nécessitant une base de temps précise.

Un 'hard-reset' sera nécessaire après de très longues périodes de mise en sommeil.

Effet de la fréquence d'horloge:

La commande NAP utilise le chien de garde interne du microcontrôleur qui est indépendant de la fréquence du résonateur.

Exemple:

```
main:
        high B.1        ; met la sortie B.1 à l'état haut
        nap 4           ; sommeil pendant 288ms
        low B.1         ; met la sortie 1 à l'état bas
        nap 7           ; sommeil pendant 2.3 s
        goto main       ; retour au point de départ
```


(c) Revolution Education Ltd.
All rights reserved

2.83 on...goto

145

Syntaxe :

ON offset GOTO adresse0, adresse1, adresseN

- Offset est une variable ou une constante, qui détermine à quelle adresse le saut aura lieu (de 0 à N)
- Les adresses sont donc les destinations du saut.

Fonction:

Dirige sur l'adresse spécifiée par offset (si celle-ci est bien comprise entre adresse0 et adresseN).

Information :

Cette commande permet de sauter à différentes adresses, selon la valeur spécifiée par offset. Si offset vaut 0, le pointeur sautera à adresse0, si offset vaut 1, le pointeur sautera à adresse1, etc ...

Si la valeur de offset dépasse le nombre maximum d'adresse spécifiée, la commande sera ignorée, le programme se poursuivra normalement.

Cette commande est équivalente à branch.

Exemple:

```
reset1:let b1 = 0
low B.0
low B.1
low B.2
low B.3

main: pause 1000
inc b1
if b1 > 3 then reset1
on b1 goto btn0,btn1, btn2, btn3
goto main

btn0: high B.0
        goto main

btn1: high B.1
        goto main

btn2: high B.2
        goto main

btn3: high B.3
        goto main
```


(c) Revolution Education Ltd.
All rights reserved

2.84 On...gosub

146

Syntaxe :

ON offset GOSUB adresse0, adresse1, adresseN

- Offset est une variable ou une constante, qui détermine à quelle adresse le saut aura lieu (de 0 à N)
- Les adresses sont donc les destinations du saut.

Fonction:

Dirige sur l'adresse spécifiée par offset (si celle-ci est bien comprise entre adresse0 et adresseN).

Information :

Cette commande permet de sauter à différentes adresses, selon la valeur spécifiée par offset. Si offset vaut 0, le pointeur sautera à adresse0, si offset vaut 1, le pointeur sautera à adresse1, etc ...

Si la valeur de offset dépasse le nombre maximum d'adresse spécifiée, la commande sera ignorée, le programme se poursuivra normalement.

La commande return de la sous-routine ramènera le pointeur sur la ligne suivant immédiatement le on gosub.

Exemple:

```
reset1:let b1 = 0
        low B.0
        low B.1
        low B.2
        low B.3

main: pause 1000
        inc b1
        if b1 > 3 then reset1
        on b1 gosub btn0,btn1, btn2, btn3
        goto main

btn0: high B.0
return

btn1: high B.1
return

btn2: high B.2
return

btn3: high B.3
return
```

2.85 output

147

Syntaxe :

Output broche, broche, broche ...

- broche(s) est (sont) une (des) variable(s) ou constante(s) qui spécifie(nt) la (les) entrée(s)-sortie(s) à utiliser.
- *Fonction:*

Passe une broche en mode sortie.

Information :

Cette commande n'est requise que pour les puces disposant du mode entrée-sortie programmable. Elle permet d'inverser le mode de fonctionnement d'une broche donnée (entrée devient sortie, ou vice et versa).

Par défaut, toutes les broches sont positionnées en mode entrée, lors de la mise sous tension (sauf, dans le cas d'une broche non programmable, et fixée en sortie) . Les broches non-programmables ne sont pas affectées par cette commande.

Ces broches sont les suivantes:

08, 08M, 08M2	0 = sortie	3 = entrée
14M2	B.0 = sortie	C.3 = entrée
18M2	C.3 = sortie	C.4, C.5 = entrées
20M2, 20X2	A.0 = sortie	C.6 = entrée
28X2, 40X2	A.4 = sortie	

Exemple:

```
main:
        input B.1       ; B1 en entrée
        reverse B.1     ; B.1 en sortie
        reverse B.1     ; B.1 en entrée
        output B.1      ; B.1 en sortie
```

2.86 owin

148

Syntaxe :

Owin broche, mode,(variable, variable ...)

- broche est une variable ou constante qui spécifie la broche à utiliser pour le raccordement au bus un fil (protocole 'One Wire')

- mode est une variable ou constante qui spécifie le mode à utiliser

Chaque bit de mode a une fonction spécifique :

Bit 0 – remise à 0 de l'impulsion envoyée avant la donnée

Bit 1 – remise à 0 de l'impulsion envoyée après la donnée

Bit 2 – mode bit (reçoit un bit, plutôt qu'un octet)

Bit 3 – force la broche à l'état haut, après réception de la donnée

Dans un but de facilité d'utilisation, les constantes pré-définies suivantes peuvent être utilisées

0	Pas de remise à 0 de OW	4	Pas de remise à 0 de OW mode bit
1	Près-remise à 0 de OW	5	Près-remise à 0 de OW mode bit
2	Remise à 0 de OW après	6	Remise à 0 de OW après mode bit
3	Remise à 0 dans les deux cas	7	Remise à 0 dans les deux cas, mode bit

- La (les) variable(s) reçoit(vent) la donnée.

- *Fonction:*

Lire une donnée (mode bit ou octet), depuis un bus type « 1 fil », connecté sur une broche entrée, avec une possibilité de remise à 0, avant ou après chaque lecture.

Cette commande ne peut être utilisée sur le matériel suivant, pour des raisons de fabrication :

20X2 C. 6 mode entrée fixe

Information :

L'utilisation du bus 1 fil est davantage détaillée, dans la documentation spécifique "Tutorial bus 1 fil". Cette commande est utilisée pour la lecture de données sur bus 1 fil.

Exemple:

```
; lecture de tempèrature depuis un DS18B20
; cette fonctionnalité est similaire à l'utilisation de la commande readtemp12
main:
     owout C.1,%1001,($CC,$44)
     ; envoie 'reset' puis 'skip ROM'
     ; puis 'convert' et applique 'pullup'

     pause 750                ; maintien le 'pullup' pendant 750 ms
     owout C.1,%0001,($CC,$BE)

     ; envoie 'reset' puis 'skip ROM'
     ; puis la commande 'read temp'
     owin C.1,%0000,(b0,b1)   ; lit et mémorise le résultat
     sertxd (#w0,CR,LF)       ; transmission série de la valeur
     goto main
```


(c) Revolution Education Ltd.
All rights reserved

2.87 owout

149

Syntaxe :

Owout broche, mode,(variable, variable ...)

- broche est une variable ou constante qui spécifie la broche à utiliser pour le raccordement au bus un fil (protocole 'One Wire')

- mode est une variable ou constante qui spécifie le mode à utiliser

Chaque bit de mode a une fonction spécifique :

Bit 0 – remise à 0 de l'impulsion envoyée avant la donnée

Bit 1 – remise à 0 de l'impulsion envoyée après la donnée

Bit 2 – mode bit (envoie un bit, plutôt qu'un octet)

Bit 3 – force la broche à l'état haut, après l'émission de la donnée

Dans un but de facilité d'utilisation, les constantes pré-définies suivantes peuvent être utilisées

0	Pas de remise à 0 de OW	4	Pas de remise à 0 de OW mode bit
1	Près-remise à 0 de OW	5	Près-remise à 0 de OW mode bit
2	Remise à 0 de OW après	6	Remise à 0 de OW après mode bit
Remise à 0 dans les deux cas		7	Remise à 0 dans les deux cas, mode bit

- - La (les) variable(s) contient(nent) la donnée à envoyer.

Fonction:

Ecrire une donnée (mode bit ou octet) sur un bus type « 1 fil », connecté sur une broche sortie, avec une possibilité de remise à 0, avant ou après chaque écriture.

Information :

L'utilisation du bus 1 fil est davantage détaillée, dans la documentation spécifique "Tutorial bus 1 fil". Cette commande est utilisée pour l'écriture de données sur bus 1 fil. Notez que certains matériels, tel le DS18B20 peuvent nécessiter un 'pullup' après l'envoi de la donnée. Cette commande ne peut être utilisée sur le matériel suivant, pour des raisons de fabrication :

20X2 C.6 = fixed input

Exemple:

```
; lecture de tempèrature depuis un DS18B20
; cette fonctionnalité est similaire à l'utilisation de la commande readtemp12
main:
     owout C.1,%1001,($CC,$44)
     ; envoie 'reset' puis 'skip ROM'
     ; puis 'convert' et applique 'pullup'
     pause 750                ; maintien le 'pullup' pendant 750 ms
     owout C.1,%0001,($CC,$BE)
     ; envoie 'reset' puis 'skip ROM'
     ; puis la commande 'read temp'
     owin C.1,%0000,(b0,b1)   ; lit et mémorise le résultat
     sertxd (#w0,CR,LF)       ; transmission série de la valeur
     goto main
```


(c) Revolution Education Ltd.
All rights reserved

2.88 pause

150

(tous les PICAXEs)

Syntaxe:

PAUSE millisecondes

- Millisecondes est une variable / constante (0-65535) qui définit le temps de pause en millisecondes (à 8 MHz sur les PICAXEs X2, et 4MHz sur les autres)

Function:

Met en pause pour une certaine durée. Cette durée a la précision de l'horloge de l'oscillateur, et suppose une fréquence de 4MHz. (8MHz pour les X2)

Information:

La commande pause crée un délai en millisecondes. La pause la plus longue est un peu plus de 65 secondes.
Pour créer une pause plus longue utiliser une boucle du type for … next.

```
for b1 = 1 to 5      ' 5 boucles
pause 60000              ' attend 60 secondes
next b1
```

Pendant une pause, la seule façon de réagir à une entrée, est l'interruption. (voir Setint)
Ne pas utiliser de longues pauses dans des boucles surveillant des changements d'entrées.
Lors de l'utilisation de pause supérieure à 5 secondes, il peut être nécessaire d'effectuer un hard reset pour charger un nouveau programme dans le microcontrôleur.
Effets d'un changement de fréquence de l'horloge du PICAXE:
La base de temps est modifiée si la fréquence par défaut est modifiée. Par exemple passer de 4 MHz à 8MHz aura pour conséquence de réduire la pause de moitié.
Lors du multitâche avec les M2 la précision de la pause est réduite en raison du fonctionnement en tâches parallèles, et la résolution minimum est de l'ordre de 20ms.
Utiliser le mono tâche si la plus grande précision est souhaitée.

Exemple:

```
main:
        high B.1                ; sortie B.1 à l'état haut
        pause 5000              ; attend 5 secondes
        low B.1                 ; sortie B.1 à l'état bas
        pause 5000              ; attend 5 secondes
        goto main               ; retour au début
```


(c) Revolution Education Ltd.
All rights reserved

2.89 pauseus

151

(tous les PICAXEs M2 + 20X2, 28X1 & X2, 40X1 & X2)

Syntax:

PAUSEUS microsecondes

- Microsecondes est une variable / constante (0-65535) qui définit le temps de pause en multiples de 10µs (à 8 MHz sur les PICAXEs X2, et 4MHz sur les autres)

Function:

Met en pause pour une certaine durée. Cette durée a la précision de l'horloge de l'oscillateur, et suppose une fréquence de 4MHz. (8MHz pour les X2)

Information:

La commande pauseus crée un délai en multiple de 10µs à 4MHz.
Comme il demande un peu de temps d'exécution, des délais courts peuvent être imprécis. Cette imprécision diminue quand le délai de pause augmente.
(NdT) Par exemple : sur un 14M2 à 4 MHz :

Pauseus 5	donne 1044 µs pour 50 théorique
Pauseus 50	donne 1533 µs pour 500 théorique
Pauseus 500	donne 6189 µs pour 5000 théorique
Pauseus 5000	donne 51235 µs pour 50000 théorique

Effets d'un changement de fréquence de l'horloge du PICAXE:
La base de temps est réduite à 5µs à 8MHz et 2.5µs à 16MHz (non-X2).

***Exe**mple:*

```
main:
        high B.1                ; sortie B.1 à l'état haut
        pauseus 5000            ; attend 50000 µs soit 50 ms
        low B.1                 ; sortie B.1 à l'état bas
        pauseus 5000            ; attend 50000 µs soit 50 ms
        goto main               ; retour au début
```


(c) Revolution Education Ltd.
All rights reserved

2.90 peek

152

Syntaxe:

PEEK localisation, variable, variable, WORD wordvariable…

- localisation est une constante ou une variable spécifiant une adresse de registre.
- variable reçoit le contenu du byte lu. On lit une variable de type word en plaçant devant le mot clé WORD.

Fonction:

Lit une donnée dans les registres RAM du microcontrôleur. Cela permet de disposer d'espaces de stockage supplémentaires non Définits par les variables bytes notés bxx (comme b11).

Information:

Pour les PICAXEs série M2 :

Il y a 512 octets de mémoire RAM. Les commandes peek et poke disposent de 256 octets de cette mémoire. Les 28 premiers octets correspondent aux bytes b0 à b27. Ces 28 premiers bytes sont accessibles soit par l'utilisation des variables bxx, soit par les commandes peek et poke.

Voir les commandes peeksfr et pokesfr pour plus de détails sur l'accessibilité aux Registres de Fonctions Spéciales (SFR= special function register).

Particularité du PICAXE 18M2: les bytes 128 à 255 de la RAM sont utilisés pour le fonctionnement du mode multitâche.

Exemple:

```
Peek 80,b1      ;envoie la valeur du registre 80 dans le byte b1
```

Pour la série X2:

Le 20X2 a 128 bytes de RAM disponibles (+128 autres pour le scratchpad)

Le 28X2 et le 40X2 ont 256 bytes de RAM disponibles (+1024 autres pour le scratchpad)

Les commandes peek et poke lisent ou écrivent dans les 256 bytes réservés à l'utilisateur. Toutefois, les 56 premiers (de 0 à 55) correspondent aux variables b0 à b55. En conséquence, ces premières variables sont accessibles de trois façons différentes, par l'appellation bxx, par les commandes peek et poke et par les variables @bptr. Les dernières variables sont seulement accessibles par les commandes peek et poke et les variables @bptr.

Voir les commandes peesfr et pokesfr pour les détails concernant l'accès aux Registres Fonctions Spéciales (SFR).

Exemple:

```
Peek 80,b1      ;envoi la valeur du registre 80 dans le byte b1
```

Pour les autres PICAXEs disponibles 28X1 et 40X1:

Les commandes peek et poke ont deux fonctions différentes:

La plus commune est la sauvegarde temporaire des variables standards (b0, b1,…) pour pouvoir les réutiliser dans des calculs.

Les adresses $50 à $7E restent libres et utilisables.

Les adresses $C0 à $EF peuvent être utilisées par le PICAXE.

La seconde façon est réservée aux utilisateurs expérimentés pour accéder aux Registres de fonctions spéciales (SFR) des microcontrôleurs.

Les adresses $00 à $1F sont des SFR qui déterminent le mode opératoire du PICAXE. Les adresses $20 à $4F et $80 à $9F sont des registres réservés à l'interpréteur basic. L'accès hasardeux à ces registres empêchera le fonctionnement de l'interpréteur.

Note : Tout cela pour dire que ces registres sont particulièrement sensibles et qu'il vaut mieux éviter d'y mettre le nez.

La commande peek peut s'appliquer à une variable type word en plaçant le mot clé WORD devant (enregistrement byte poids faible en tête).

Exemple:

```
   Peek 80,b1          ;envoie la valeur du registre 80 dans la variable b1
Peek 80, word w1       ;envoie la valeur des registres 80 et 81 dans la variable w1
                       ;poids faible en tête (b0 en 80 et b1 en 81)
```


(c) Revolution Education Ltd.
All rights reserved

2.91 peeksfr

154

Syntaxe:

PEEKSFR localisation, variable, variable,…

- localisation est une constante ou une variable spécifiant une adresse de registre (les valeurs entre 0 et 255 ne sont pas toutes utilisées)
- variable est une variable type byte où la valeur est retournée.

Fonction:

Cette commande lit une donnée à partir des Registres de Fonctions Spéciales (SFR), elle permet aux utilisateurs expérimentés de lire les réglages des périphériques intégrés au microcontrôleur. Elle ne concerne que la série M2 et X2, pour les autres, voir la commande peek.

Information:

La commande peeksfr est réservée aux utilisateurs expérimentés pour étudier les registres de fonctions spéciales (SFR).

Seules les SFR associées à des périphériques (par ex: ADC ou timers) sont accessibles.

L'utilisation des commandes peek ou poke sur des SFR associées à des opérations du programme PICAXE (ex: FSR, EEPROM ou table de registres) entrainera un reset immédiat.

Série X2

Comme les adresses sont limitées aux valeurs comprises entre 0 et 255, les notices Microchip ne mentionnent pas le "F" initial des valeurs hexadécimales (par ex: BAUDCON FB8h devient $B8).

Série M2

Comme les adresses sont limitées aux valeurs comprises entre 0 et 255, ces adresses sont composées dans les notices Microchip selon le format suivant:

Bit 7 à 5 => Banc mémoires de $00 à $07

Bit 4 à 0 => Adresse de $0C à $1F dans ce banc ($00 à $0B sont invalides et provoquent un reset immédiat).

Ex: BAUDCON, adresse 01Fh dans le banc 3 devient %011 1111

Exemple:

```
Peeksfr $9B,b1  ;envoie la valeur du registre OSCTUNE dans la variable b1
```


(c) Revolution Education Ltd.
All rights reserved

2.92 play

155

Syntaxe:

PLAY broche, musique ***(tout PICAXE avec plus de 8 broches)***

PLAY broche,musique, masque_LED ***(PICAXE série M2 seulement)***

Play musique, Option_LED ***(PICAXE 08M2 seulement)***

- broche est une constante ou une variable spécifiant l'E/S utilisée (pas pour le 08M2)
- musique est une constante ou une variable (0 à 3) spécifiant la musique à jouer

  0 – Happy Birthday

  1 – Jingle Bells

  2 – Silent Night

  3 – Rudolph the Red Nosed Reindeer
- Masque_LED (PICAXE M2 sauf 08M2) est une constante ou une variable spécifiant les ports alimentant les flashs (leds) pendant la musique. Ex: %00000011 fait flasher les ports 0 et 1.
- Option_LED (08M2 seulement) est une constante ou une variable (0 à 3) spécifiant les ports alimentant les flashs pendant la musique sur le port 2

  0 – Pas de flash sur les sorties

  1 – Flash sur la sortie C.0

  2 – Flash sur la sortie C.4

  3 – Flashes sur les sorties C.0 et C.4, alternativement

Fonction:

Joue une musique préenregistrée sur une des sorties.

Description:

Les PICAXEs peuvent jouer de morceaux de musique. La commande play permet de jouer quatre musiques préenregistrées plutôt destinés à de petits programmes simples. La commande "tune" permet de générer des musiques personnalisées au format RTTTL.

Voir les circuits d'adaptation des piezzo, haut-parleur ou amplis dans la commande tune.

Effets de la vitesses d'horloge:

Cette commande adapte automatiquement la vitesse à 4 MHz.

Exemple:

```
; PICAXE 08M2
Play 3,1                        ;joue la musique n°3 et flash sur la sortie C.0

; PICAXEs série M2
Play B.3,1,%00000011        ;joue la musique n°3 sur B.3, plus flashes sur B.0 et B.1

; autres PICAXEs
Play 2,1                        ;joue la musique n°1 sur la sortie 2
```

2.93 poke

156

syntaxe:

POKE localisation, data, data, WORD wordvariable…

- localisation est une constante ou une variable spécifiant une adresse de registre.
- data est la valeur du byte à écrire dans le registre. On écrit une variable de type word en plaçant devant le mot clé WORD.

Fonction:

Ecrire une donnée localisée par le registre FSR. Cela permet d'écrire dans des registres non Définits par b0,b1,…

Pour les PICAXEs série M2 :

Il y a 512 octets de mémoire RAM. Les commandes peek et poke disposent de 256 octets de cette mémoire. Les 28 premiers octets correspondent aux bytes b0 à b27. Ces 28 premiers bytes sont accessibles soit par l'utilisation des variables bxx, soit par les commandes peek et poke.

Voir les commandes peeksfr et pokesfr pour plus de détails sur l'accessibilité aux Registres de Fonctions Spéciales (SFR= special function register).

Particularité du PICAXE 18M2: les bytes 128 à 255 de la RAM sont utilisés pour le fonctionnement du mode multitâche.

Exemple:

```
Poke 80,b1      ;écrit la valeur de la variable b1 dans le registre 80
```

Pour la série X2:

Le 20X2 a 128 bytes de RAM disponibles (+128 autres pour le scratchpad)

Le 28X2 et le 40X2 ont 256 bytes de RAM disponibles (+1024 autres pour le scratchpad)

Les commandes peek et poke lisent ou écrivent dans les 256 bytes réservés à l'utilisateur. Toutefois, les 56 premiers (de 0 à 55) correspondent aux variables b0 à b55. En conséquence, ces premières variables sont accessibles de trois façons différentes, par l'appellation bxx, par les commandes peek et poke et par les variables @bptr. Les dernières variables sont seulement accessibles par les commandes peek et poke et les variables @bptr.

Voir les commandes peesfr et pokesfr pour les détails concernant l'accès aux Registres Fonctions Spéciales (SFR).

Exemple:

```
poke 80,b1      ; écrit la valeur de la variable b1 dans le registre 80
```

Pour les autres PICAXEs disponibles 28X1 et 40X1:

Les commandes peek et poke ont deux fonctions différentes:

La plus commune est la sauvegarde temporaire des variables standards (b0, b1,…) pour pouvoir les réutiliser dans des calculs.

Les adresses $50 à $7E restent libres et utilisables.

Les adresses $C0 à $EF peuvent être utilisées par le PICAXE.

La seconde façon est réservée aux utilisateurs expérimentés pour accéder aux Registres de fonctions spéciales (SFR) des microcontrôleurs.

Les adresses $00 à $1F sont des SFR qui déterminent le mode opératoire du PICAXE. Les adresses $20 à $4F et $80 à $9F sont des registres réservés à l'interpréteur basic. L'accès hasardeux à ces registres empêchera le fonctionnement de l'interpréteur.

Note : Tout cela pour dire que ces registres sont particulièrement sensibles et qu'il vaut mieux éviter d'y mettre le nez.

La commande poke peut s'appliquer à une variable type word en plaçant le mot clé WORD devant (enregistrement byte poids faible en tête).

Exemple:

```
   poke 80,b1              ;sauve la valeur de b1 dans le registre 80
Poke 80, word w1           ;sauve la valeur de la variable w1 dans les registres 80 et 81
                           ;poids faible en tête (b0 en 80 et b1 en 81)
```


(c) Revolution Education Ltd.
All rights reserved

2.94 pokesfr

158

syntaxe:

POKESFR, localisation, data, data,…

- localisation est une constante ou une variable spécifiant une adresse de registre (les valeurs entre 0 et 255 ne sont pas toutes utilisées)
- variable est une variable type byte contenant la valeur à enregistrer.

Fonction:

Cette commande écrit des données dans les Registres de Fonctions Spéciales (SFR), elle permet aux utilisateurs expérimentés d'ajuster les réglages des périphériques intégrés au microcontrôleur. Elle ne concerne que la série M2 et X2, pour les autres, voir la commande poke.

Information:

La commande pokesfr est réservée aux utilisateurs expérimentés pour modifier les registres de fonctions spéciales (SFR).

Seules les SFR associées à des périphériques (par ex: ADC ou timers) sont accessibles.

L'utilisation des commandes peek ou poke sur des SFR associées à des opérations du programme PICAXE (ex: FSR, EEPROM ou table de registres) entrainera un reset immédiat.

Série X2

Comme les adresses sont limitées aux valeurs comprises entre 0 et 255, les notices Microchip ne mentionnent pas le "F" initial des valeurs hexadécimales (par ex: BAUDCON FB8h devient $B8).

Série M2

Comme les adresses sont limitées aux valeurs comprises entre 0 et 255, ces adresses sont composées dans les notices Microchip selon le format suivant:

Bit 7 à 5 => Banc mémoires de $00 à $07

Bit 4 à 0 => Adresse de $0C à $1F dans ce banc ($00 à $0B sont invalides et provoquent un reset immédiat).

Ex: BAUDCON, adresse 01Fh dans le banc 3 devient %011 1111

Exemple:

```
Pokesfr $9B,b1    ;envoie la valeur de la variable b1 dans le registre OSCTUNE
```


(c) Revolution Education Ltd.
All rights reserved

2.95 pullup

159

Syntaxe:

PULLUP masque

PULLUP OFF (=PULLUP 0)

PULLUP ON (=PULLUP 255)

- Masque est une variable ou une constante spécifiant les entrées activées

Fonction:

Active ou désactive la résistance de pullup sur le port spécifié. La résistance de pullup relie l'entrée sélectionnée au + alimentation.

Information:

La commande pullup active ou désactive la résistance de pullup sur certaines entrées. Toutes les broches n'ont pas une résistance de pullup. Si une broche est configurée en sortie, la résistance de pullup est automatiquement désactivée.

La résistance de pullup interne permet par exemple de supprimer la résistance externe lors de l'utilisation d'interrupteur et de simplifier le circuit.

La fonction "masque" dépend du type de PICAXE utilisé et peut utiliser jusqu'à 16 bits, de bit0 à bit16. La présence ou l'absence de résistance de pullup sur une broche dépend de la configuration interne du microcontrôleur.

08M12	Bit0-bit4 = C.0 à C.4	
14M2	Bit0-bit4 = B.0 à B.7	Bit8-bit15 = C.0 à C.7
18M2	Bit0-bit4 = B.0 à B.7	
20M2	Bit0-bit4 = B.0 à B.7	Bit8-bit15 = C.0 à C.7
20X2	Bit0-bit7 = C.0, C.6, C.7, B.0, B.1, B.5, B.6, B.7	
28X2/40X2	Bit0-bit7 = B.0 à B.7	
28X2-5v/40X2-5v	On = Tous les PORTB	
28X2-3v/40X2-3V	Bit0-bit4 = B.0 à B.7	

Sur les anciens PICAXEs 28X2-5V / 40X2-5V, les résistances de pullup sont uniquement sur les ports B et ne peuvent être activées individuellement. L'utilisation de "ON" ou "OFF" active les 8 résistances de pullup en même temps.

Exemple:

```
Pullup on                  ; active les pullups sur PICAXE 28X2-5V
Pullup %11110000           ; active les pullups de B.4 à B.7 sur un 28X2
Pullup %00000111           ; active les pullups sur C.0, C.6, C.7 d'un 20X2
Pullup %0000100000000100   ; active les pullups sur B.2 et C.3 d'un 20M2
```

2.96 pulsin

160

Syntaxe:

PULSIN broche, état, wordvariable

- Broche est une constante ou une variable spécifiant l'entrée utilisée.
- État est une constante ou une variable (0 ou 1) spécifiant le sens du front de départ de la mesure. L'unité de mesure est de 10µs à la fréquence de 4MHz.
- Wordvariable est une variable de type word recevant le résultat (de 1 à 65535). Si cette valeur dépasse le maximum de 65535 (timeout), le résultat est 0.

Fonction:

Mesure la longueur d'une impulsion sur une entrée.

Information:

La commande pulsin mesure la longueur d'une impulsion.

En absence d'impulsions dans la fenêtre de mesure, le résultat est 0.

Si état = 1 , la mesure est initialisée par un front montant (de 0 à 1), si état = 0, la mesure est initialisée par un front descendant (de 1 à 0).

Utilisez la commande count pour compter un nombre d'impulsions dans un temps donné.

Il est normal d'utiliser un variable de type word avec cette commande.

Effet de la fréquence d'horloge:

Fréquence horloge	Unité de mesure	Temps maximum (timeout)
4 MHz	10µs	0,65536 s
8 MHz	5 µs	0,32768 s
16 MHz	2,5 µs	0,16384 s
32 MHz	1,25 µs	0,08192 s
64 MHZ	0,625 µs	0,04096 s

Exemple:

```
Pulsing C.3, 1, w1        ;enregistre dans w1,la longueur d'une impulsion positive sur C.3
```


(c) Revolution Education Ltd.
All rights reserved

2.97 pulsout

161

Syntaxe:

PULSOUT broche, temps

- broche est une constante ou une variable spécifiant la sortie utilisée.
- temps est une constante ou une variable spécifiant la longueur de l'impulsion (de 0 à 65535) en dizaines de µs (à la fréquence de 4 MHz).

Fonction:

La commande pulsout génère une impulsion de longueur donnée.

Information:

L'état préalable de la sortie Définit le sens de l'impulsion. Si la sortie est basse (0), l'impulsion sera positive, si la sortie est haute (1), elle sera négative.

La commande Définit automatiquement la broche utilisée comme une sortie. Par sécurité, il est préférable de s'en assurer.

Effet de la fréquence horloge:

Fréquence horloge	Temps par unité
4 M	10µs
8 MHz	5 µs
16 MHz	2,5 µs
32 MHz	1,25µs
64 MHz	0,625 µs

Exemple:

```
....
Low B.1
Pulsout B.1,150          ;génère une impulsion positive
                         ;de 1,50 ms sur B.1 (à 4 MHz)
....
```


(c) Revolution Education Ltd.
All rights reserved

2.98 put

162

syntaxe:

PUT localisation, data, data, WORD wordvariable….

- Localisation est une constant ou une variable spécifiant une adresse du scratchpad. Les valeurs valides sont:

  0 à 127 pour la série X1

  0 à 127 pour les PICAXEs 20X2

  0 à 1023 pour les autres PICAXEs de la série X2

- Data est une constant ou une variable contenant la valeur à sauvegarder. Pour les variables de type word, placer le mot clé WORD devant.

Fonction:

Ecrit des données dans un emplacement du scratchpad.

Information:

La fonction des commandes put et get est d'enregister temporairement des données dans la mémoire scratchpad du microcontrôleur. Cela permet de libérer des variables standards (b0, b1,…) pour les réutiliser dans des calculs.

Put et get n'ont pas d'effet sur le pointeur d'adresse du scratchpad (ptr) qui reste inchangé par l'utilisation de ces commandes.

Lorsque des variables de type sont utilisées, les deux bytes de ces variables sont enregistrés byte poids faible en tête.

Exemple:

```
Put 1, b1              ;sauve la valeur de b1 dans le registre 1
Put 1, word w0   ;sauve la valeur de w0 dans deux registres : b0 en 1 et b1 en 2
```

2.99 pwm

163

Syntaxe:

PWM broche, duty, cycles

- Broche est une constante ou une variable spécifiant l' E/S utilisée.
- Duty est une constante ou une variable (0 à 255) spécifiant le niveau analogique.
- Cycles est une constante ou une variable (0 à 255) spécifiant le nombre d'impulsions. Chaque cycle dure environ 5 ms à la fréquence horloge de 4 MHz.

Fonction:

Génère un train d'impulsions sur une sortie qui redevient ensuite une entrée, (potentiel flottant)

Information:

Cette commande est ancienne et rarement utilisée actuellement. Pour les contrôles moteurs (et autres) par pwm, la commande pwmout est recommandée.

Cette commande pwm produit des trains d'impulsions pour générer une sortie pseudo analogique. Cette fonction est améliorée par un filtre R/C lissant la sortie. La commande doit être renouvelée périodiquement pour entretenir la tension analogique produite.

Exemple:

```
Debut:
    Pwm C.4, 150, 20   ;génère un train d'impulsions sur la broche C.4
    Pause 20           ;pause 20ms (à 4 MHz)
    Goto debut         ;boucle sur debut
```


(c) Revolution Education Ltd.
All rights reserved

2.100 pwmduty

164

Syntaxe:

PWMDUTY broche, rapport_cyclique

- Broche est la sortie utilisée par la commande pwmout correspondante.

  Les sorties ne sont pas toutes compatibles avec PWM, voir le diagramme des sorties du PICAXE.

- Rapport_cyclique est une constante ou une variable (0 à 1023) spécifiant le créneau positif du cycle / période;

Fonction:

Modifie le rapport cycle d'une commande pwmout en fonction et utilisant la même broche.

Information:

Sur certains PICAXEs, la commande pwmduty peut être utilisée pour modifier le rapport cyclique d'une sortie pwm, sans réinitialiser le timer interne (comme c'est le cas avec la commande pwmout).

Information:

Voir la commande pwmout pour plus de détails.

Exemple:

```
; pour une fréquence horloge de 4 MHz
        Pwmout C.2,200, 400 ;signal pwm 5000 Hz, rapport cyclique 50%
        Pwmout C.2,200, 600 ;signal pwm 5000 Hz, rapport cyclique 75 %
        Pwmout C.2,200, 200 ;signal pwm 5000 Hz, rapport cyclique 25 %
```

Exemple:

```
Init:
        Pwmout C.2, 150, 100        ;génère une sortie pwm sur la broche C.2
Do                                  ;début de boucle
        Pwmduty C.2, 150            ;augmente du rapport cyclique
        Pause 1000                  ;pause 1 s
        Pwmduty C.2, 50             ;diminue le rapport cyclique
        Pause 1000                  ;pause 1 s
Loop                                ;fin de boucle Do
```


(c) Revolution Education Ltd.
All rights reserved

2.101 pwmout

165

Syntaxe:

PWMOUT, broche, période, rapport_cyclique

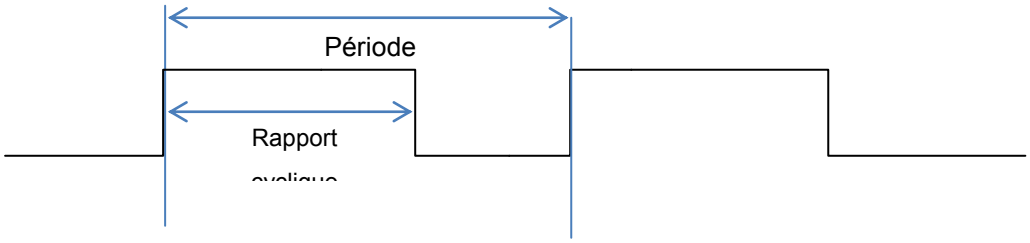
PWMOUT,PWMDIV4, broche, période, rapport_cyclique

PWMOUT,PWMDIV16, broche, période, rapport_cyclique

PWMOUT,PWMDIV64, broche, période, rapport_cyclique

PWMOUT, broche, OFF

- Broche est une constante ou une variable spécifiant la sortie utilisée. Toutes les sorties ne sont pas compatibles PWM, voir le diagramme des sorties du PICAXE.
- Période est une constante ou une variable (0 à 255) définissant la période (fréquence) du PWM.
- Rapport_cyclique est une constante ou une variable (0 à 1023) définissant le créneau positif du cycle.



Le mot clé PWMDIV adapte la fréquence horloge à la fréquence du PWM.

Fonction:

Génère une sortie PWM continue utilisant le module PWM interne au microcontrôleur. Voir aussi la commande HPWM , équivalente à la commande pwmout sur des sorties différentes.

Information:

Cette commande est différente de la plupart des commandes basic, le signal PWM est généré en continu (en tache de fond) tant qu'une autre commande ne le modifie pas, elle est utilisée pour commander des moteurs à courant continu à différentes vitesses. L'arrêt est obtenu par la commande "pwmout, broche, OFF" (= pwmout,broche,0, 0).

Période PWM = (période+1) x 4 x vitesse horloge (à 4 MHz, vitesse horloge = 1 / 4000000)

Rapport cyclique PWM = (rapport cyclique) x vitesse horloge

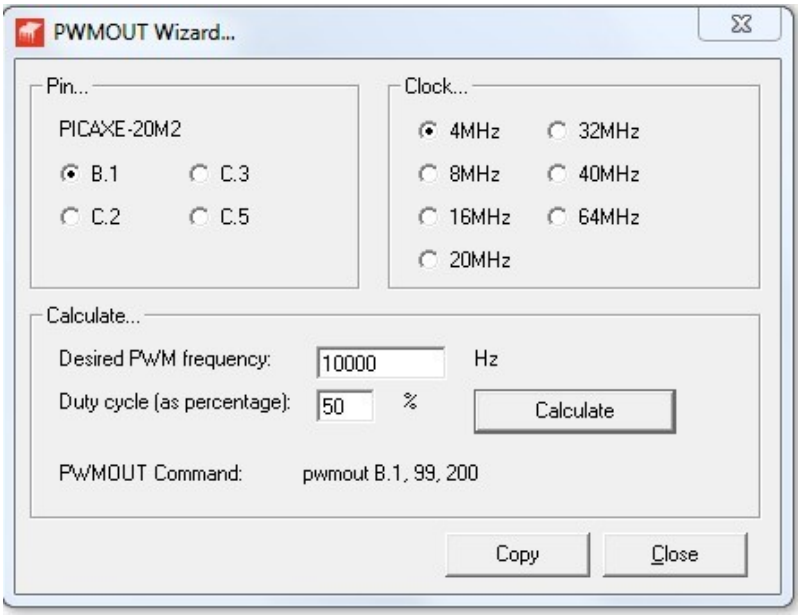
Note: La période et le rapport cyclique sont liés à la fréquence horloge. Pour un même signal, la modification de la fréquence horloge entraine une modification de la commande.

Dans tous les cas: Fréquence PWM = 1 / période PWM

Un utilitaire du logiciel Programming Editor permet de générer automatiquement cette commande à partir des paramètres désirés:

Sélectionnez: PICAXE > Wizard > pwmout dans les onglets de P.E.

La fenêtre ci contre apparait:



Les broches compatibles avec le PICAXE sont dans la cadre "Pin".

Sélectionnez la broche choisie.

Sélectionnez la bonne fréquence horloge.

Réglez la fréquence et le rapport cyclique (en %) recherchés.

Cliquez sur "Calculate" et la commande **pwmout B.1,99,200** apparait

En cliquant sur "Copy", la ligne s'écrit automatiquement dans le programme à l'emplacement du curseur…simplement.

Comme la commande pwmout utilise un module interne au microcontrôleur, il y a certaines restrictions:

1. La commande ne fonctionne que sur certaines broches
2. Le rapport cyclique est codé sur 10 bits (0 à 1023). Le temps "rapport cyclique" ne peut pas être plus long que la période du signal. Réglage maximum = 4 x période. Un dépassement provoquera un comportement erratique.
3. Le module PWM utilise le même timer pour les broches C.1 et C.2 sur les PICAXEs de 28 ou 40 broches. La fréquence PWM sur ces broches doit être identique, mais les rapports cycliques peuvent être différents.
4. Généralement, la commande "servo" utilise le même timer que "pwmout", ces commandes ne peuvent donc pas être utilisées en même temps (voir * ci-dessous).
5. Le signal pwm est interrompu pendant les commandes "nap" et "sleep" et s'arrête après "end".
6. Pwmout 1 peut être utilisé en même temps que hpwm.
7. Pwmout 2 ne peut pas être utilisé en même temps que hpwm (voir 3 ci-dessus).
8. Pwmout dépend de la fréquence horloge. Certaines commandes, comme readtemp, réduisent automatiquement cette fréquence à 4 MHz, cette éventualité doit être prise en compte.

(*) Pour palier le partage d'un même timer entre les commandes "servo" et pwmout, la dernière génération de PICAXEs intègre un timer dédié permettant l'utilisation de ces commandes en même temps.

Cela s'applique suivant ce tableau:

PICAXE	Broches indépendantes	Broches partagées par timer servo
14 M2	B.2 , B.4	C.0 , C.2
18 M2	B.3 , B.6	
20 M2	B.1 , C.2	C.3 , C.5
28 M2	B.0 , B.5	C.1 , C.2

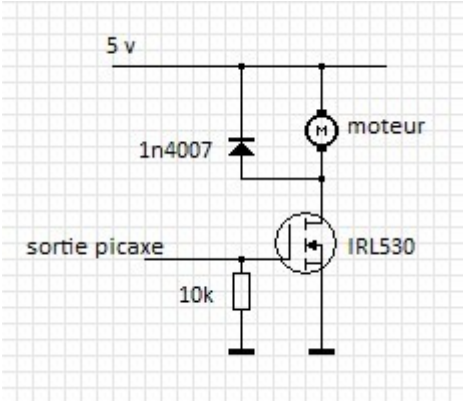
Sur les PICAXE de la série X2 (seulement) toute commande "pwmout" fait passer les commandes pwmdiv4 et pwmdiv16 à pwmdiv1.

Pour maintenir ces commandes après une nouvelle commande pwmout, il faut les rééditer par une commande:

PWMOUT PWMDIV4, broche , pour chaque broche utilisée.

Sur les pixaxes partageant un timer (ex: C.1 et C.2 sur un 28X2), la commande "pwmout, broche, OFF stoppera les deux canaux. Pour stopper un seul canal, il faut utiliser "pwmduty broche, 0"

La commande pwmout initialise la sortie pour la fonction pwm et réinitialise également les timers internes, il est donc préférable d'utiliser la commande pwmduty pour modifier les rapports cycliques rapidement.



Conduite moteur par PWMOUT:

Pendant l'initialisation de la commande pwmout, la tension sur la broche du PICAXE est instable (flottante).

Il est essentiel de fixer cette tension à 0 v par une résistance de 10 kΩ pour éviter tout fonctionnement erratique du moteur.

Exemple:

```
Init:
      Pwmout C.2, 150, 100      ;génère un signal continu sur C.2
Do                              ;début boucle Do/loop
      Pwmduty C.2, 150          ;augmente le rapport cyclique
      Pause 1000                ;pause 1 s
      Pwmduty C.2,50            ;diminue le rapport cyclique
      Pause 1000                ;pause 1 s
Loop                            ; fin boucle do/loop
```

2.102 random

168

Syntaxe:

RANDOM wordvariable

- Wordvariable est à la fois "l'espace de travail " et le résultat. Comme la commande random génère une suite peudo-aléatoire, il est préférable de l'utiliser dans une boucle. Wordvariable doit être de type word, une variable de type byte donnera de mauvais résultat

Fonction:

Génère un nombre pseudo aléatoire dans wordvariable

Description:

La commande random génère un nombre pseudo aléatoire compris entre 0 et 65535. Les microcontrôleurs utilisent des algorithmes pour générer des séquences de nombres aléatoires et les séquences ne sont jamais réellement aléatoires.

Comme sur les ordinateurs, il faut essayer de modifier l'initialisation du calcul en utilisant une variable "évolutive" comme time sur la série M2 ou timer sur la série X2. Sinon, en supposant que la valeur initiale de w0 soit 0, random w0 donnera un nombre N1 (0 à 65535), ce nombre servira de base pour calculer le nombre suivant N2 et ainsi de suite, donc toute séquence initialisée avec 0 donnera: 0, N1,N2…

Cette recherche de l'initialisation aléatoire n'est à faire qu'une fois, Exemple pour la série M2:

```
w0=time                    ; initialisation de w0 avec la variable système time
random w0          ; La valeur w0 dépendra de la valeur de time
```

Un autre moyen, utilisable sur tous les PICAXEs, est de profiter d'une boucle (ex: boucle d'attente pour l'appui sur un switch) dans laquelle on introduit la commande random.

```
Do                         ;début boucle DO
ramdom w0                  ;modification répétitive de random
loop while switch=0
```

Si un nombre aléatoire de type byte (0 à 255) est suffisant, il est possible de décomposer la variable word en ses deux bytes ex: w0 = (b1,b0) , une variable aléatoire de type word donne ainsi deux variables aléatoires de type byte.

2.103 read

169

Syntaxe:

READ localisation, variable, variable ,WORD wordvariable, …

- localisation est une constante ou une variable spécifiant l'adresse du byte lu
- variable reçoit le contenu du byte lu. On lit une variable de type word en plaçant devant le mot clé WORD.

Fonction:

La commande read lit une donnée enregistrée en mémoire EEPROM. Les données sauvegardées en EEPROM sont conservées lorsque le microcontrôleur n'est plus sous tension. La sauvegarde se fait par la commande WRITE.

En fait, dans le système PICAXE, une variable de type word est la concaténation de deux variables de type byte ex:w0=(b1,b0).

Il faut donc lire deux bytes consécutifs pour reconstituer une variable type word. Avec le mot clé WORD l'enregistrement des bytes se fait dans l'ordre (b0,b1), "poids faible en tête".

Pour les PICAXEs 08, 08M, 08M2, 14M, 18, 18M et 18M2, la mémoire EEPROM est partagée avec la mémoire du programme. Voir la commande EEPROM (DATA) pour plus de détails.

Exemple:

```
Debut:
  For b0=0 to 10                ;boucle for/next
      Read b0,b1                ;Enregistre la valeur localisée par b0 dans le byte b1
      Serout B.7,N2400,(b1)     ;transmission vers le LCD série
  Next b0                       ;boucle valeur suivante de b0, jusqu'à 10
```

2.104 readadc

170

(tous les PICAXEs)

Syntaxe:

READADC canal,variable

- canal est une variable constant définissant la broche ADC.
- Variable prends la valeur de la donnée lue (type byte).

Function:

Lit le canal analogique (ADC) sur 8 bits de résolution et l'affecte à variable.
Sur les PICAXEs X2 la commande adcsetup doit être utilisée pour configurer la broche en entrée analogique.
Sur les autres PICAXEs, la configuration est automatique.

Information:

La commande readadc est utilisée pour lire la valeur analogique présente sur une broche d'entrée du microcontrôleur
Noter que toutes les entrées ne possèdent pas un convertisseur analogique-numérique. (ADC)
Voir le schéma de brochage du PICAXE utilisé.

Exemple:

```
main:
 readadc C.1,b1              ; Affecte à b1 la valeur lue sur C.1
 if b1 > 50 then flsh              ; si b1 > 50 saute au sous-programme flsh
 goto main                   ; sinon, reboucle au début
flsh:
      high B.1               ; met la sortie B.1 à l'état haut
      pause 5000             ; pause de  5 secondes
      low B.1                ; met la sortie B.1 à l'état bas
      goto main              ; retour au programme principal
```


(c) Revolution Education Ltd.
All rights reserved

2.105 readadc10

171

(tous les PICAXEs sauf 08, 18, 18A, 18M, 28A)

Syntaxe:

READADC10 canal, wordvariable

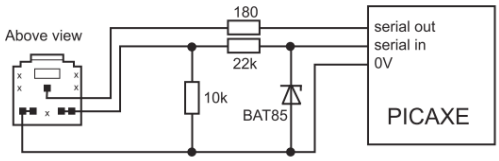
- canal est une variable constant définissant la broche ADC.
- Variable prends la valeur de la donnée lue (type word).

Fonction:

Lit le canal analogique (ADC) sur 10 bits de résolution et l'affecte à wordvariable.
Sur les PICAXEs X2 la commande adcsetup doit être utilisée pour configurer la broche en entrée analogique.
Sur les autres PICAXEs, la configuration est automatique.
Sur les PICAXEs X2 on doit utiliser le canal ADC, et non le N° de broche dans la commande readadc.
Par exemple **readadc10 0,w1** mais non **readadc10 A.0,w1**

Information:

La commande readadc10 est utilisée pour lire la valeur analogique présente sur une broche d'entrée du microcontrôleur avec une résolution de 10bits (0-1023).
Noter que toutes les entrées ne possèdent pas un convertisseur analogique-numérique (ADC).
Voir le schéma de brochage du PICAXE utilisé.
Comme le résultat est sur 10 bits une variable type word doit être utilisée. Pour une variable du type byte, utiliser la commande **readadc**
Pour les utilisateurs des anciens câbles série AXE026 (ne s'applique pas aux câbles USB AXE027):
Quand on utilise la commande **debug** pour disposer de la valeur sur 10 bits, la liaison électrique via le câble série peut affecter les valeurs lues. Dans ce cas, il est recommandé d'utiliser un circuit modifié sur la liaison série. La diode Schottky résoud ce problème.



Exemple:

```
main:
 readadc10 C.1,w1          ; Affecte à w1 la valeur lue sur C.1
 debug                     ; transmet les valeurs à l'ordinateur
 pause 200           ; courte pause de 200ms
 goto main           ; retour au début
```

2.106 readdac

172

(PICAXEs 08M2, 14M2, 18M2, 20M2, 28X2, 40X2)

Syntax:

READDAC variable

- variable est une variable byte de la valeur DAC

Function:

Lit une valeur DAC dans la variable.

Information:

La commande readdac lit le niveau en cours de DAC qui doit avoir été configuré par les commandes **dacsetup** and **daclevel.** On peut le considérer comme étant la lecture analogique du niveau de tension de DAC.

Exemple:

```
main:
readdac b1      ; lit le niveau  DAC dans b1.
```


(c) Revolution Education Ltd.
All rights reserved

2.107 readdac10

171

(Tous les PICAXEs sauf 08, 18, 18A, 28, 28A, 28X, 40X)

Syntaxe:

READDAC10 wordvariable

- variable est une variable word de la valeur DAC

Fonction:

Lit une valeur DAC dans la variable.

Information:

La commande readdac lit le niveau en cours de DAC qui doit avoir été configuré par les commandes **dacsetup** and **daclevel.** On peut le considérer comme étant la lecture analogique **readadc10** du niveau de tension de DAC.

Exemple:

```
main:
readaddac10 w1    ; lit le niveau  DAC dans w1.
```

2.108 readi2c

174

Cette commande est obsolète, pensez à utiliser la commande hi2cin.

Syntaxe:

READI2C (variable,…)

READI2C location, (variable,…)

- variable(s) dans laquelle la lecture de l'octet du périphérique i2c se positionne.
- location est une variable/constante spécifiant l'octet (ou le word) de départ de la lecture.

Fonction:

Permet de lire les données du périphérique i2c et de la positionner dans une variable (ou plusieurs variables).

Information:

Cette commande est utilisée pour lire des données d'un périphérique i2c.

Les données sont lues à partir de l'adresse de départ indiquée, la lecture des autres données séquentielles est possible si le périphérique i2c le supporte.

Si les périphériques i2c sont mal configurés ou que des données i2cslave erronées ont été utilisées, la valeur 255 ($FF) est chargée dans chaque variable.

Exemple:

```
; Utilisation d'un module horloge DS1307
; ce module utilise le format BCD (voir utilisation de BCDTOASCII et BINTOASCII)
; le PICAXE est maître, le module DS1307 est l'esclave et utilise l'adresse i2c %11010000

      i2cslave, %11010000, i2cslow, i2cbyte   ; adressage DS1307, mode
                                              ;slow en octet
debut:
      readi2c 0,(b0,b1,b2,b3,b4,b5,b6,b7)           ; lecture et affichage de
                                              ;l'heure et de la date
      debug                   ; affichage des valeurs sur l'écran de l'ordinateur
      pause 2000              ; attente de 2 secondes
      goto debut              ; aller à debut, bouclage du programme
```

2.109 readinternaltemp

175

(PICAXEs 08M2, 14M2, 18M2+, 20M2)

Syntaxe:

READINTERNALTEMP voltage, offset, variable

READINTERNALTEMP voltage, - offset, variable

- Voltage est une constante indiquant la tension d'alimentation. Les options sont:

IT_5V0	5V d'alimentation
IT_4V5	4.5V d'alimentation
IT_4V0	4V d'alimentation
IT_3V5	3.5V d'alimentation
IT_3V3	3.3V d'alimentation
IT_3V0	3V d'alimentation
IT_RAW_H	Lecture de valeur brute (réglage haut, au delà de 4V)
IT_RAW_L	Lecture de valeur brute (réglage bas, any voltage)

- Offset facteur de correction optionnel, à 0 par défaut
- Variable reçoit la valeur de la température.

Fonction:

La commande readinternaltemp mesure la chute de tension analogique au travers de 2 (low) ou 44 (high) diodes internes. Cela donne une indication approximative de la température.

Information:

Cette commande est utilisée pour indiquer la température interne du chip. Il est destiné à fournir, en comparant avec un seuil, une indication de défaut de refroidissement du composant, et non un capteur précis de température!
Pour des mesures précises, utiliser plutôt un capteur DS18B20 et la commande **readtemp**.
De façon interne, une mesure analogique (ADC) est faite de la tension de deux diodes connectées à la tension d'alimentation.
La lecture varie avec la température. Comme la référence de la mesure ADC est la tension d'alimentation, la lecture varie aussi en fonction de cette alimentation. (par exemple baisse de tension de piles ou batterie)
Quand **IT_RAW_H** ou **IT_RAW_L** sont utilisées, la valeur brute est charge dans une variable word. Offset est ignoré dans ce cas, et doit être mis à 0.
Quand d'autres réglages sont utilisés le PICAXE transforme mathématiquement cette valeur en degrés Celsius avec approximation. Si nécessaire, une valeur d'offset peut être ajoutée ou soustraite de la valeur brute avant la conversion afin d'améliorer la précision.
Bien noter que ce principe ne peut en aucun cas être un capteur précis, et doit être uniquement utilisé comme indicateur de température limite. Le seuil et l'offset varient selon les composants.
Pour la précision, utilisez un DS18B20 !

***Exe**mple:*

```
main:
readinternaltemp IT_5V0,0,b1
debug
pause 500
goto main
```

Informations complémentaires:

Les équations mathématiques utilisées pour convertir les valeurs brutes en degrés Celsius sont:
5V0 RAW_H +/- K -508 * 14 / 13 + 5
4V5 RAW_H +/- K -450 * 14 / 15 + 5
4V0 RAW_H +/- K -378 * 14 / 18 + 5
3V5 RAW_L +/- K -668 * 14 / 10 + 5
3V3 RAW_L +/- K -647 * 14 / 10 + 5
3V0 RAW_L +/- K -609 * 14 / 10 + 5

2.110 readfirmware

177 (PICAXE 20X2, 28X2, 40X2)

Syntax:

READFIRMWARE variable

- variable est une variable type byte qui reçoit la donnée révision

Function:

Affecte le numéro de révision du firmware du bootstrap PICAXE à variable.

Information:

La commande **readfirmware** extrait numéro de révision du firmware du bootstrap version et l'affecte à variable.

Ne pas confondre la révision (programme utilisateur) avec la version de firmware (bootstrap PICAXE).

Exemple:

```
main:
        readfirmware b1  ; Affecte le numéro de révision à b1
```

2.111 readmem

(ne concerne que le 28A obsolète ; non traduit)

(c) Revolution Education Ltd.
All rights reserved

2.112 readtable

179 (PICAXE 14M2, 18M2, 20M2, 20X2, 28X1, 28X2, 40X1, 40X2)

Syntaxe:

readtable position,variable

- Position est une variable /constante spécifiant l'adresse.
- variable reçoit la valeur byte stockée à l'adresse position de la table.

Fonction:

Lit la valeur à une adresse donnée de la table de recherche.

Information:

Quelques PICAXEs permettent à des données (par ex. messages LCD) d'être définites dans une table à l'intérieur du programme, via la commande **table.**

C'est une façon très efficace de stockage de données.

Voir la commande 'table' pour plus de détails.

Des blocs de données peuvent également être transférés en RAM via la commande **tablecopy.**

Exemple:

```
TABLE 0,("Hello World")   ; sauve des données dans une table
main:
for b0 = 0 to 10          ; débute une boucle
  readtable b0,b1               ; lit la donnée dans la table .
  serout B.7,N2400,(b1)   ; et la transmet au module LCD série
next b0                         ; caractère suivant.
```


(c) Revolution Education Ltd.
All rights reserved

2.113 readoutputs

180 (Tous PICAXE sauf 08, 28X2, 40X2)

Syntaxe:

READOUTPUTS variable

- variable est une variable byte recevant la valeur des sorties

Fonction:

Affecte la valeur de output pins à variable.

Information:

L'état des broches de sortie peut être affecté à une variable en utilisant la commande **readoutputs**.

Noter que c'est différent de '**let var = pins**', qui lit l'état des entrées, et non des sorties.

Cette commande n'est normalement pas utilisée avec les M2, X1 or X2 dont l'état des sorties peut être lu directement par '**let var = outpinsX**'

Exemple:

```
main:
        readoutputs b1   ; Affecte à b1 l'état des sorties.
```

2.114 readportc

181 `(PICAXE 28X1, 40X1)`

Syntaxe:

READPORTC variable

- variable est une variable byte recevant la valeur de portc

Fonction:

Affecte la valeur de portc à variable.

Information:

L'état des broches du portc sur le 40X1 peut etre affecté à une variable en utilisant la commande **readportc**. Cette commande n'est pas requise sur les autres PICAXEs, la commande '**let var = pinsC**' pouvant être utilisée.

Exemple:

```
main:
readportc b1 ; Affecte à b1 l'état du port C
```


(c) Revolution Education Ltd.
All rights reserved

2.115 readrevision

182 (PICAXE 20X2, 28X2, 40X2)

Syntaxe:

READREVISION variable

- variable est une variable byte recevant la donnée de révision.

Fonction:

Affecte la valeur de la révision de programme à variable.

Information:

En utilisant la directive #revision il est possible d'incorporer la donnée de révision du code utilisateur dans le programme téléchargé. La commande **readrevision** l'affecte à une variable.

La donnée de révision est utilisée également par la commande **booti2c** . Ne pas confondre la révision (programme utilisateur) avec la version de firmware (version du bootstap du PICAXE)

Exemple:

```
main:
        readrevision b1 ; Affecte à b1 la donnée de révision
```


(c) Revolution Education Ltd.
All rights reserved

2.116 readsilicon

183 (PICAXE 20X2, 28X2, 40X2)

Syntaxe:

READSILICON variable

- variable est une variable recevant siliconvalue

Fonction:

Affecte la valeur de siliconrevision d'un X2 à variable.

Bits 7 - 5 PICAXE Type

000	reserved for future use
001	PICAXE-20X2 (PIC18F14K22)
010	PICAXE-28X2-5V (PIC18F2520)
011	PICAXE-40X2-5V (PIC18F4520)
100	PICAXE-28X2 (PIC18F25K22)
101	PICAXE-40X2 (PIC18F45K22)
110	PICAXE-28X2-3V (PIC18F25K20)
111	PICAXE-40X2-3V (PIC18F45K20)

Bits 4 - 0

Version de puce Microchip

Information:

La commande readsilicon extrait l'information concernant la puce silicium du microcontrôleur et l'affecte à une variable.

Ne pas confondre avec la révision (programme utilisateur) ou la version de firmware (version de bootstrap du PICAXE).

Exemple:

```
main:
readsilicon b1   ; affecte la version de puce  à b1
```

2.117 readtemp

184 (PICAXE 14M2, 18M2, 20M2, 20X2, 28X1, 28X2, 40X1, 40X2)

Syntaxe:

READTEMP pin,variable

- Pin est la broche d'entrée.
- Variable reçoit la donnée lue (byte).

Fonction:

Lit la température d'un capteur numérique DS18B20 et l'affecte à variable.

La conversion prend jusquà 750 ms. Readtemp réalise une conversion sur 12 bits et tronque le résultat pour une valeur entière de degrés. (Contrairement à la doc Rev Ed qui parle d'arrondi à la valeur entière la plus proche - NdT)

Pour la valeur sur 12bits, utiliser la commande **readtemp12**.

Information:

La température est renvoyée en degrés, et le capteur fonctionne de -55 to + 125 degrés Celsius.

Noter que le bit 7 est 0 pour des températures positives, et 1 pour les températures négatives.

(C'est-à-dire que les valeurs négatives apparaîtront comme 128 + valeur de la température)

readtemp ne fonctionne pas avec les anciens DS1820 ou DS18S20, car ils ont une résolution interne différente .

Cette commande n'est pas à utiliser avec le mode d'alimentation fantôme du DS18B20 , le 5V du capteur devant toujours être connecté.

Cette commande ne peut être utilisée sur les broches suivantes compte tenu des restrictions des puces silicium.

08, 08M, 08M2 C.3,C. 5 = entrée figée, C.0 = sortie figée

14M, 14M2 C.3 = entrée figée, B.0 = sortie figée

18M2 C.4, C.5 = entrée figée

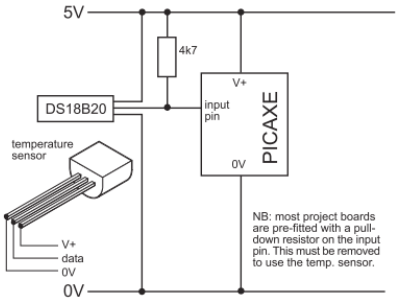
20M, 20M2, 20X2 C.6 = entrée figée

Effets d'accroissement de la fréquence d'horloge::

Cette commande ne fonctionne qu'à 4MHz. Les M2, X1 et X2 utilisent automatiquement l'oscillateur interne 4 MHz pour cette commande.

Exemple:

```
main:
readtemp C.1,b1          ; lit la valeur et l'affecte à b1
if b1 > 127 then neg     ; test si valeur négative
serout B.7,N2400,(#b1)   ; transmet la valeur au LCD série
goto loop
neg:
let b1 = b1 - 128        ; calcule la valeur négative
serout B.7,N2400,("-")   ; transmet le signe - au LCD série
serout B.7,N2400,(#b1)   ; transmet la valeur au LCD série
goto main
```



(c) Revolution Education Ltd.
All rights reserved

2.118 readtemp12

185 (PICAXE 14M2, 18M2, 20M2, 20X2, 28X1, 28X2, 40X1, 40X2)

Syntaxe:

READTEMP12 pin,wordvariable

- Pin est la broche d'entrée.
- Wordvariable reçoit la donnée lue (type word).

Fonction:

Lit la température d'un capteur numérique DS18B20 et l'affecte à wordvariable.

La conversion prend jusquà 750 ms.

readtemp et **readtemp12** mettent le même temps de conversion.

Information:

Cette commande est pour les utilisateurs avancés. Pour des valeurs entières de degrés, utiliser la commande **readtemp**.

La température est renvoyée en valeur brute sur 12 bits dans une variable word. (résolution de 0.0625 Degré). L'utilisateur doit interpréter le résultat en faisant un calcul.

Voir la datasheet du DS18B20 pour plus d'informations sur la relation Température/donnée brute.

Voir le circuit électrique sur la commande readtemp.

Readtemp12 ne fonctionne pas avec les anciens DS1820 ou DS18S20, car ils ont une résolution interne différente .

Cette commande n'est pas à utiliser avec le mode d'alimentation fantôme du DS18B20 , le 5V du capteur devant toujours être connecté.

Cette commande ne peut être utilisée sur les broches suivantes compte tenu des restrictions des puces silicium.

08, 08M, 08M2 3 = entrée figée

14M, 14M2 C.3 = entrée figée

18M2 C.4, C.5 = entrée figée

20M,20M2, 20X2 C.6 = entrée figée

Effets d'accroissement de la fréquence d'horloge::

Cette commande ne fonctionne qu'à 4MHz. Les M2, X1 et X2 utilisent automatiquement l'oscillateur interne 4 MHz pour cette commande.

Exemple:

```
main:
readtemp12 1,w1          ; lit la valeur et l'affecte à w1
debug                    ; transmet sur l'écran de l'ordinateur.
goto main
```

2.119 readowclk

186

Obsolète

(c) Revolution Education Ltd.
All rights reserved

2.120 resetowclk

187

Obsolète

(c) Revolution Education Ltd.
All rights reserved

2.121 Readowsn

188 (PICAXE 14M2, 18M2, 20M2, 20X2, 28X1, 28X2, 40X1, 40X2)

Syntaxe:

readowsn pin

- Pin est une variable/constante qui précise l'entrée utilisée..

Fonction:

Lit un numéro de série d'un composant 1-wire Dallas/Maxim.

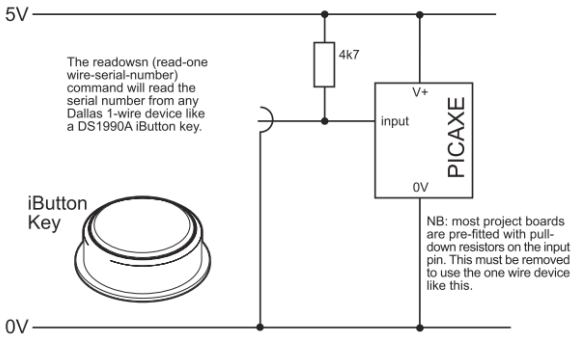
Information:

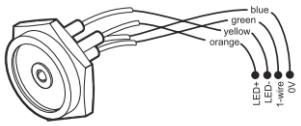
Cette commande (read-one-wire-serial-number) lit le numéro de série unique d'un composant 1-wire Dallas/Maxim (par ex: capteur numérique de T° DS18B20, horloge DS2415, ou iButton DS1990A).

Dans le cas d'un iButton (DS1990A) Ce numéro de série est gravé au laser sur le boitier du iButton.

La commande **readowsn** lit le numéro de série et place le code famille dans b6, le numéro de série dans b7 à b12, et la checksum dans b13 .

A noter qu'il ne faut pas utiliser par ailleurs les variables b6 à b13 (ni w3 à w6)





Part RSA002 - iButton Contact probe

NB : La plupart des platines projets sont prééquipés d'une résistance de tirage (pull down) entre l'entrée et le 0V. Cette résistance doit être enlevée dans ce cas d'utilisation.

Cette commande ne peut être utilisée sur les broches suivantes compte tenu des restrictions des puces silicium.

08, 08M, 08M2 3 = fixed input

14M, 14M2 C.3 = fixed input

18M2 C.4, C.5 = fixed input

20M,20M2, 20X2 C.6 = fixed input

Exemple:

```
main:
let b6 = 0              ; remise à 0 du code famille
```

```
loop1:              ; boucle jusqu'à ce que le code soit différent de 0
readowsn C.2
if b6 = 0 then loop1
; Fait ici un simple contrôle
; la valeur de b12 ne peut être FF
; Si c'est le cas, cela signifie que le composant
; à été enlevé avant la fin de la lecture,
; ou qu'il y a eu court-circuit.
if b12 = $FF then main
; Tout est OK et on continue…
debug               ; ok on affiche à l'écran
pause 1000          ; petite pause
goto main
```


(c) Revolution Education Ltd.
All rights reserved

2.122 reconnect

190

Syntaxe :

RECONNECT

Fonction:

Réinitialise un PICAXE déconnecté, afin de lui permettre d'accepter un nouveau téléchargement.

Information :

Les puces PICAXE vérifient en permanence si une liaison PC est en train de tenter de leur envoyetr un nouveau programme.

Il peut arriver que l'utilisateur souhaite désactiver cette fonctionnalité afin d'utiliser cette broche en communication série (commande serrxd).Une fois la déconnection effective, il ne sera plus possible de procéder à un téléchargement, tant que :

1) Une commande reconnect ne soit utilisée

2) Une commande reset ne soit utilisée

3) Un reset matériel ne soit effectué

Un reset matériel, permettra, dans tous les cas, de procéder à un nouveau téléchargement.

Exemple:

```
disconnect
serrxd [1000, timeout],@ptrinc,@ptrinc,@ptr
reconnect
```


(c) Revolution Education Ltd.
All rights reserved

2.123 reset

191

Syntaxe :

Reset

Fonction:

Force une remise à zéro. Cette commande à le même effet que l'appui sur le bouton reset, ou, d'une coupure temporaire de l'alimentation.

Information :

Le pointeur est ramené à la première ligne, les variables sont remise à zéro, les pointeurs de piles ré-initialisés.

Exemple:

```
main:
        let b2 = 15             ; détermine la valeur de b2
        pause 2000              ; attente 2 secondes
        gosub flsh              ; appelle sous-procedure flsh
        let b2 = 5              ; détermine la valeur de b2
        pause 2000              ; attente 2 secondes
        reset                   ; re-démarrage
```


(c) Revolution Education Ltd.
All rights reserved

2.124 restart

192

Syntaxe :

Restart tâche

- tâche est une variable ou constante (SANS accentuation) qui spécifie le nom de la tâche à re-démarrer.

Fonction:

Re-démarre une tâche.

Information :

Les puces de type M2 peuvent faire fonctionner un certain nombre de tâches en parallèle. La commande restart re-démarre la tâche à sa première ligne, ou à l'endroit de son interruption, si elle a été suspendue à cet endroit. Cette commande n'affecte pas les variables, pour les remettre à zéro, utilisez plutôt la commande reset.

Exemple:

```
start0:
        b3 = 0                  ; remet à 0 b3
        loop0:
        high B.0                ; B.0 à 1
        pause 1000              ; attente 1 seconde
        low B.0                 ; B.0 à 0
        pause 1000              ; attente 1 seconde
        inc b3                  ; incrèmente b3
        goto loop0              ; boucle


start1:
        inc b4                  ; incrèmente b4
        if b4 > 10 then         ; si b4 > 10 alors
                restart 0       ; re-démarre la tâche 0.  b3 repasse à 0
                b4 = 0
        end if
        debug                   ; affichage variables
        pause 1000
        goto start1
```


(c) Revolution Education Ltd.
All rights reserved

2.125 resume

193

Syntaxe :

Resume tâche

- tâche est une variable ou constante (SANS accentuation) qui spécifie le nom de la tâche à relancer.

Fonction:

Relance une tâche à son point de suspension.

Information :

Les puces de type M2 peuvent faire fonctionner un certain nombre de tâche en parallèle. La commande resume re-démarre la tâche à l'endroit de son interruption, les autres tâche ne sont pas affectées. Si la tâche était déjà en cours (non suspendue), la commande sera ignorée.

Exemple:

```
start0:
        high B.0                ; B.0 à 1
        pause 100               ; attente 0.1 seconde
        low B.0                 ; B.0 à 0
        pause 100               ; attente 0.1 seconde
        goto start0             ; boucle

start1:
        pause 5000              ; attente 5 secondes
        suspend 0               ; suspension de la tâche 0
        pause 5000              ; attente 5 secondes
        resume 0                ; relance la tâche 0
        goto start1             ; boucle
```


(c) Revolution Education Ltd.
All rights reserved

2.126 return

194

Syntaxe :

RETURN

Fonction:

Sortie de sous routine, et retour juste après l'instruction d'appel.

Information :

Cette commande doit exclusivement être employée après l'utilisation d'un gosub correspondant.

Dans le cas contraire, le programme crashera.

Exemple:

```
main:
     let b2 = 15          ; définit b2
     pause 2000           ; attente 2 secondes
     gosub flsh           ; appel sous-routine flsh
     let b2 = 5           ; définit b2
     pause 2000           ; attente 2 secondes
     gosub flsh           ; appel sous-routine flsh
end                       ; stop en cas de plantage accidentel dans sous-routine

flsh:
     for b0 = 1 to b2     ; boucle 2 fois
     high B.1             ; passe B.1 à 1
     pause 500            ; attente 0.5 seconde
     low B.1              ; passe B.1 à 0
     pause 500            ; attente 0.5 seconde
     next b0              ; fin de boucle
return                    ; fin de la sous-procédure, et retour
```


(c) Revolution Education Ltd.
All rights reserved

2.127 reverse

195

Syntaxe :

Reverse broche, broche, broche ...

- Broche est une variable ou constante qui spécifie la ou les broche(s) concernée(s)

Fonction:

Inverse le sens de fonctionnement d'une ou plusieurs broches (entrée devient sortie, et vice et versa)

Information :

Cette commande n'est utilisable qu'avec les puces disposant de broches programmables. Elle peut être utilisée pour transformer une broche pré-programmée en entrée, en broche de sortie.

Toutes les broches sont, par défaut, des entrées, lors de la mise sous tension (saut pour les broches "figées" en mode sortie)

Cette commande n'affecte pas les broches non programmables. Ces dernières sont :

08,08M, 08M2	0 = sortie fixe	0 = entrée fixe
14M2	B.0 = sortie fixe	C.3 = entrée fixe
18M2	C.3 = sortie fixe	C.4, C.5 = entrée fixe
20M2, 20X2	A.0 = sortie fixe	C.6 = entrée fixe
28X2, 40X2	A.4 = sortie fixe	

Exemple:

```
main:
        input B.1       ; passe B.1 en mode entrée
        reverse B.1     ; passe B.1 en mode sortie
        reverse B.1     ; passe B.1 en mode entrée
        output B.1      ; passe B.1 en mode sortie
```


(c) Revolution Education Ltd.
All rights reserved

2.128 rfin

196

Syntaxe :

Rfin broche, variable, variable, variable, variable , variable, variable, variable, variable

- Broche est une variable ou constante qui spécifie la broche concernée
- Variables sont les huit octets de données

Fonction:

Réception, sur huit octets, d'une donnée codée type Manchester, en provenance, soit d'un module NKM2401, soit d'un autre PICAXE, utilisant la commande rfout, et un module de transmission HF

Il est à noter, que cette commande requiert l'utilisation de huit octets précisément.

Information :

Cette commande reçoit et décode un train de huit octets, transmis par liaison HF, et reçu par un module genre 'NKM2401', ou transmit depuis un autre PICAXE. Ce protocole de transmission assure une meilleure fiabilité que l'utilisation de la fonction serout, et permet l'utilisation de modules radio à bas prix.

Attention, cette command est dite 'bloquante' : aucune autre action n'aura lieu tant que rfin sera en attente des huit octets de données.

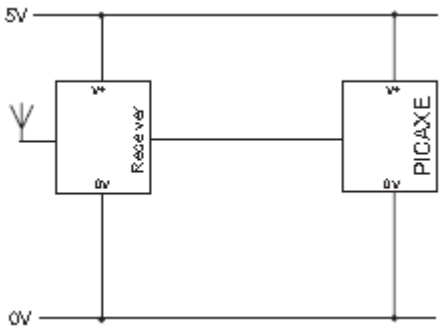
Si l'on souhaite ne pas bloquer le déroulement du programme, durant l'attente de réception de données, l'emploi du module NKM2401 en récepteur dédicacé, est impératif. Le PICAXE, pourra alors, à son gré, récupérer les données stockées dans le NKM2401.

Ce module est utilisable avec tout type de PICAXE, y compris ceux ne gérant pas la commande rfin, dans la mesure où la communication PICAXE-NKM01 est de type série. Pour plus d'information sur ce module, reportez-vous à la documentation AXE213, disponible ici :

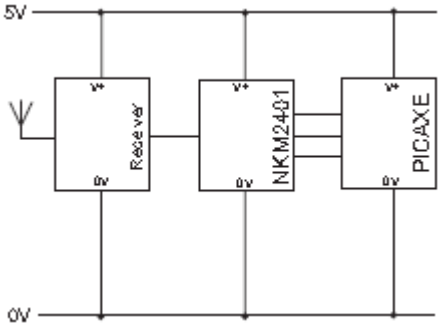
www.rev-ed.co.uk/docs/axe213.pdf

Vous y trouverez également les notices d'emplois des modules HF économiques.

Utilisation de la commande rfin (bloquante)



Utilisation du module NKM2401 (non bloquant)

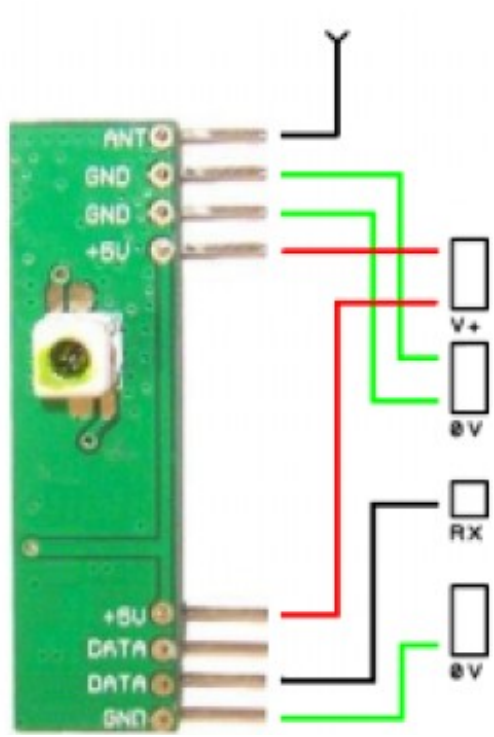


Exemple de câblage

La sortie de données du module récepteur (un RFA001, dans cet exemple)

est directement connectée à une broche entrée du PICAXE. Notez que

l'utilisation d'une antenne adaptée est impérative, et que la distance entre l'émetteur et le récepteur doit être d'au moins un mètre.



Effet de l'augmentation de la vitesse d'horloge:

Cette commande ne fonctionne qu'à 4 Mhz. Les puces de

série X2 et M2 basculent automatiquement à cette fréquence.

```
main:
        rfin C.0, b0,b1,b2,b3,b4,b5,b6,b7
        debug
        goto main
```


(c) Revolution Education Ltd.
All rights reserved

2.129 rfout

198

Syntaxe :

Rfout broche, (donnée, donnée, donnée, donnée, donnée, donnée, donnée, donnée)

- Broche est une variable ou constante qui spécifie la broche concernée
- Données représentent les huit octets à transmettre (variables ou constantes)

Fonction:

Emission, sur huit octets, d'une donnée codée type Manchester, destiné, soit à un module NKM2401, soit à un autre PICAXE, utilisant la commande rfin, et un module de transmission HF

Il est à noter, que cette commande requiert l'utilisation de huit octets précisément.

Information :

Cette commande code et emet un train de huit octets, transmis par liaison HF, à destination un module genre 'NKM2401', ou d'un autre PICAXE. Ce protocole de transmission assure une meilleure fiabilité que l'utilisation de la commande serout, et permet l'utilisation de modules radio à bas prix.

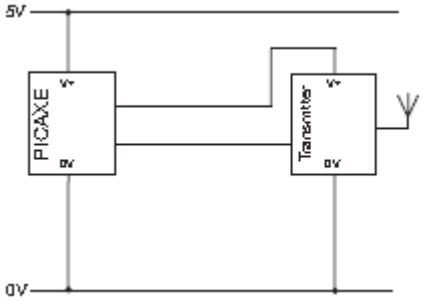
Cette commande effectue le même codage qu'un module NKM01, mais ce dernier permet l'utilisation d'un PICAXE ne gérant pas la commande rfout.

Pour plus d'information sur ce module, reportez-vous à la documentation AXE213, disponible ici :

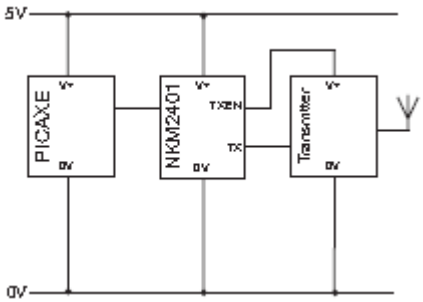
www.rev-ed.co.uk/docs/axe213.pdf

Vous y trouverez également les notices d'emplois des modules HF économiques.

Utilisation de la commande rfout



Utilisation de la commande serout avec module NKM2401



(c) Revolution Education Ltd.
All rights reserved

Exemple de câblage

L'entrée de données du module émetteur (un RFA001, dans cet exemple

NdT erreur, à mon sens, même ref que le RX)

Est directement connectée à une broche sortie (TX) du PICAXE.

Une autre broche est dévolue à la validation de l'émission(TXEN)

en alimentant l'émetteur, si nécessaire. Attention, si cette alimentation

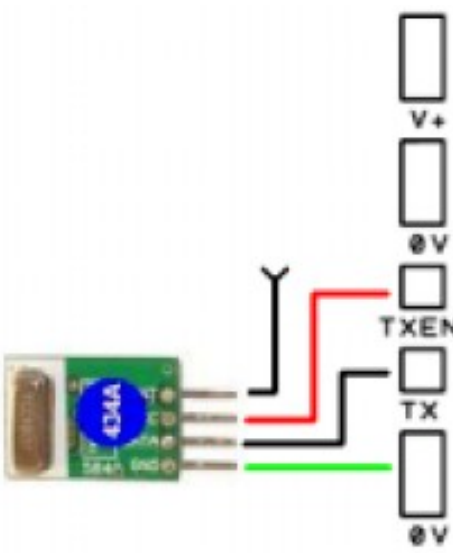
requiert plus de 20mA, l'utilisation d'un transistor de commande

est impérative.

Ne pas laisser l'émetteur sous tension en permanence.

Ne pas utiliser de tampons de sortie type Darlington,pour les données,

le module doit être directement connecté au PICAXE



Effet de l'augmentation de la vitesse d'horloge:

Cette commande ne fonctionne qu'à 4 Mhz. Les puces de

série X2 et M2 basculent automatiquement à cette fréquence

```
main:
        readtemp C.1, b7                ; lecture température et stockage dans b7
        bintoascii b7,b8,b9,b10         ; conversion en 3 caractères ASCII
        high b.1                        ; alimentation module radio (TXEN)
        rfout b.0,("Temp=",b8,b9,b10)   ; envoie donnée (TX)
        low b.1                         ; arrêt module radio (TXEN)
        pause 2000                      ; attente 2 secondes
        goto main                       ; boucle permanente
```


(c) Revolution Education Ltd.
All rights reserved

2.130 run

199

Syntaxe :

RUN slot

- Slot, qui doit être considéré un numéro est une variable, ou constante, spécifiant quel numéro de programme lancer.

Fonction:

Lance un autre numéro de programme.

Information :

Les puces type 28X2 et 40X2 ont la possibilité de gérer jusqu'à quatre programmes différents, identifiés par un numéro. Par défaut, le numéro zéro est toujours lancé au démarage. Les puces type 20X2 ne supportent que le numéro zéro.

Un nouveau programme peut être téléchargé, sous un numéro choisi, par le biais de la directive #slot, qui rajoutera automatiquement une ligne au programme. Il n'est possible de télécharger qu'un seul programme numéroté à la fois. Les autres programmes ne sont pas affectés par ce téléchargement.

Pour démarer un deuxième programme (après l'avoir téléchargé, avec la directive #slot 1), la commande 'run 1'doit être lancée. Cette commande arrêtera le programme en cours, et lancera immédiatement le deuxième programme.Les variables, et les états des broches ne seront pas affectés, et pourront donc être partagés d'un programme à l'autre. Cependant, toutes les autres fonctions système telles que états des piles gosub/return seront remise à zéro, lors du démarage du deuxième programme. Ainsi, l'accès au programme numéro un, depuis le programme numéro 0 peut être considéré comme un 'goto autre programme', et non comme un 'gosub'

Une fois le programme numéro 1 lancé, vous pouvez, au sein de ce programme relancer le programme précédent, via l'instruction 'run 0'. Si vous souhaitez également remettre les variables à zéro, il vous faudra alors utiliser la commande 'reset'. Ceci équivaudra alors à un 'run 0' avec remise à zéro des variables.

Rappellez-vous, que dans le cas du re-téléchargement d'un nouveau programme, celui-ci sera par défaut téléchargé en numéro 0. Si vous souhaitez télécharger ce programme en numéro 2, vous devrez utiliser la directive '#slot 2' dans le programme.

Toutes les puces type X2 peuvent également exécuter un programme stocké dans une EEPROM type i2c. Ne seront alors reconnus comme programmes que ceux numérotés de 4 à 7 (sur une EEPROM d'adresse 000). Il est possible d'utiliser jusqu'à huit EEPROM d'adressages différents, ce qui, théoriquement permet l'usage de 32 programmes (8 X 4). Si l'adresse de l'EEPROM utilisée n'est pas 000, les bits 5 et 7 du numéro de programme seront utilisés comme adresse EEPROM, ainsi, pour lancer un programme numéro 5, stocké dans une EEPROM d'adresse A2 à 0, A1 à 1 et A0 à 1, la commande sera :

Run %0011xx101 (xx ne comptent pas)

Certaines restrictions s'appliquent, lors du lancement de programmes stockés en EEPROM :

1) les broches SDA et SCL de la liaison i2c sont réservées. Le bus i2c ne peut être utilisé par ailleurs.
2) Le déroulement du programme sera notablement ralentit. Il est plus long d'accéder aux info's stockées en EEPROM.

Voir également la commande 'booti2c', qui peut être préférable a l'utilisation des programme numéroté de 4 à 7.

Information additionnelles : Comprendre la notion de numéro de programme ('slot'que l'on peut aussi considérer comme 'position' ou 'emplacement'–NdT-)

Les puces de type X2 disposent de la possibilité de gérer quatre programmes distincts, numérotés (par 'slot') de 0 à 3. Ces programmes sont complètements indépendants (NdT: sauf au niveau des variables, voir plus haut). Lors de la mise sous tension, ou d'un reset, le programme 0 (slot 0) est automatiquement lancé. Les autres programmes peuvent alors être lancés, via l'instruction 'run'.

Chaque nouveau téléchargement, par défaut, porte le numéro 0. Pour télécharger un programme à une autre position (autre numéro, donc autre 'slot'), utilisez la directive #slot. Exemple :

```
#slot 1
```

Ce qui téléchargera le programme en position1 et non 0. Les autres programmes (numéros différents) ne seront pas affectés.

Vous remarquerez, qu'après le téléchargement, le programme 0 sera toujours lancé, quel que soit le dernier programme téléchargé. Si vous souhaitez le démarage du dernier programme téléchargé (le 1, par exemple), placez, dans le programme initial (slot 0) la commande 'run 1'.

Dans la mesure où le PICAXE ne dispose que d'une zone de données EEPROM interne (utilisée par les commande read et write), tous les téléchargements, quel que soit le slot, utiliseront cette même EEPROM . Pour éviter cela, utilisez une directive #no_data pour le téléchargement. Ceci évitera la modification des données contenues en EEPROM. (les commandes écriture en EEPROM seront ignorées).

La méthode habituelle d'utilisation des numéros de programmes consiste à tester une entrée (par exemple, la présence d'un jumper) lors d'un reset, et de "brancher" sur un programme spécifique, suivant l'entrée concernée.

```
#slot 0
if pinC.1 = 1 then
        run 1
endif
if pinC.2 = then
        run 2
endif
```

Cependant, les programmes différemment numérotés, peuvent être combinés, au sein d'un 'programme plus long', en tenant compte des points suivants :

1) Aucun **gosub** ne peut être partagé, entre programmes de numéros différents (y compris procédure d'interruption)
2) La pile de contrôle des gosub/return est remise à zéro, à chaque changement de programme.
3) La zone scratchpad n'est pas modifiée
4) La commande '**run X**' doit être considérée comme un **goto** vers un autre programme.

Il est à noter, que l'usage de la commande '**run 0**' n'a pas le même effet que la commande '**reset**', qui, elle, remettra les variables à zéro, et ré-initialisera les broches entrées-sorties.

Numéros de programmes extérieurs à la puce

En sus de l'utilisation de l'espace interne de la puce, pour le stockage de programmes, il est possible d'utiliser une EEPROM externe (24LC128), connectée sous i2c, pour stocker 4 programmes supplémentaires. Il est possible d'utiliser jusqu'à 8 de ces EEPROM simultanément, sur le même bus i2c, ce qui donne théoriquement la possibilité d'implanter 32 programmes différents (8 X 4).

Avec une 24LC128 adressée à 0 (broches A0, A1 et A2 à 0) les numéros de slot transmis via i2c seront simplement de 4 à 7.

Si l'on rajoute d'autres 24CL128, les numéros de programmes (slot) seront calculés comme suit :

Bit7	28LC128 adresse broche A2
Bit6	24LC128 adresse broche A1
Bit5	24LC128 adresse broche A0
Bit4, Bit3	Réservés pour une utilisation future, mettre à zéro
Bit2	1=I2C, 0 = interne
Bit1, 0	les quatre numéros possibles de programmes.

Faire fonctionner un programme stocké en EEPROM impose un certain nombre de limitations :

1) Le bus i2c est strictement résérvé à la lecture du programme.
2) Les broches i2c ne peuvent être utilisées pour d'autres fonctions.
3) N'importe quel autre matériel commandé par i2c/spi sera complétement ignoré
4) La vitesse d'exécution du programme sera réduite, le temps d'accès à la mémoire EEPROM extérieure étant plus long.

Les commandes EEPROM habituelles (read write) continueront de n'avoir d'effet que sur l'EEPROM interne au picbasic. Pas sur la (les) EEPROM extérieure(s)..

Exemple:

```
#slot 0
init:
        if pinC.1 =1 then main      ; teste une broche entrée, après reset
        run 1                       ; si l'entrée est à zéro, lance le prog 1
main: high B.1                      ; procédure normale, le 1 tourne.
```

2.131 select case \ case \ else \ endselect

203

Syntaxe:

SELECT VAR

CASE VALEUR

{code}

CASE VALEUR,VALEUR,...

{code}

CASE VALEUR à VALEUR

{code}

CASE ?? VALEUR

{code}

ELSE

{code}

ENDSELECT

VAR est la valeur à tester
VALEUR est une variable ou une constante
?? peut être une des conditions suivantes :

= (égal à / est égal à)

<> (n' est pas égal à)

> (plus grand que)

>= (plus grand ou égal à)

< (plus petit que)

<= (plus petit ou égal à)

Concerne :

08 / 08M / 08M2

14M / 14M2

18 / 18A / 18M / 18M2 / 18X

20M / 20M2 / 20X2

28A / 28X / 28X1 / 28X2

40X / 40X1 / 40X2

Commentaire :

Compare la valeur VAR avec VALEUR. Si le test est positif (si la condition est remplie), la section {code} est exécutée

Les tests sont réalisés successivement.

Si le test est négatif (la condition n' est pas remplie) le test suivant est exécuté.

Si un test est positif (la condition est remplie), la section {code} qui suit est exécutée puis le programme saute à ENDSELECT (fin du test).

Si aucun test n'est positif (aucune condition n' est remplie), c' est la commande ELSE qui est exécutée.

Exemple:

```
select case b1
        case 1          ;est-ce que b1=1?
                high 1  ;si oui
        case 2,3        ;sinon, est-ce que b1=2 ou b1=3
                low 1   ;si oui
        case 4 to 6     ;si non, est-ce que b1 est compris entre 4 et 6 inclus?
                high 2  ;si oui
        else            ;si non
                low 2
endselect               ;fin des tests
```


(c) Revolution Education Ltd.
All rights reserved

2.132 serin

204

Syntaxe:

SERIN pin, baudmode, (qualifier, qualifier…)

SERIN pin, baudmode, (qualifier, qualifier…), {#}variable, {#}variable…

SERIN pin, baudmode, {#}variable, {#}variable…

Options de syntaxe indiquant un délai optionnel pour les séries M2, X1 et X2:

SERIN [timeout], pin, baudmode, (qualifier…)

SERIN [timeout], pin, baudmode, (qualifier…), {#}variable, {#}variable

SERIN [timeout], pin, baudmode, {#}variable, {#}variable

SERIN [timeout, address], pin, baudmode, (qualifier…)

SERIN [timeout, address], pin, baudmode, (qualifier…), {#}variable, {#}variable…

SERIN [timeout, address], pin, baudmode, {#}variable, {#}variable…

- pin indique le numéro de la broche d'entrée utilisée par le PICAXE.
- baudmode est une variable/constante (de 0 à 7) qui indique le mode de fonctionnement :

Txxx indique le mode normal Nxxx indique le mode inversé

Vitesse de transmission possible pour la série 08 / 08M / 18 / 18A / 28 / 28A :

4MHz	8MHz	16MHz
T300_4	T600_8	T1200_16
T600_4	T1200_8	T2400_16
T1200_4	T2400_8	T4800_16
T2400_4	T4800_8	T9600_16
N300_4	N600_8	N1200_16
N600_4	N1200_8	N2400_16
N1200_4	N2400_8	N4800_16
N2400_4	N4800_8	N9600_16

Vitesse de transmission possible pour les autres séries :

4MHz	8MHz	16MHz	32 MHz	64MHz
T600_4	T1200_8	T2400_16	T4800_32	T9600_64
T1200_4	T2400_8	T4800_16	T9600_32	T19200_64
T2400_4	T4800_8	T9600_16	T19200_32	T38400_64
T4800_4	T9600_8	T19200_16	T38400_32	T76800_64
N600_4	N1200_8	N2400_16	N4800_32	N9600_64
N1200_4	N2400_8	N4800_16	N9600_32	N19200_64
N2400_4	N4800_8	N9600_16	N19200_32	N38400_64
N4800_4	N9600_8	N19200_16	N38400_32	N76800_64

- qualifier est une variable/constante (de 0 à 255) qui doit être reçu avant la prise en compte des octets suivants
- timeout permet de choisir le temps d'attente d'une réception (en milliseconde)
- address correspond au nom de label vers lequel le programme est dérouté si le temps d'attente est dépassé
- variable est la variable dans laquelle on positionne la donnée reçue (de 0 à 127)

(c) Revolution Education Ltd.
All rights reserved

Fonction:

Permet de recevoir une donnée en format série sur une broche d'entrée du PICAXE (pas de parité, 8 bits de données, 1 bit d'arrêt).

Information:

Cette commande ne doit pas utiliser la broche de téléchargement qui demande l'utilisation de la commande serrxd

Si vous utilisez une interface simple à résistances, il faut choisir le mode N (mode inversé).

Si vous utilisez une interface avec un circuit MAX232, il faut choisir le mode T.

La vitesse de réception du PICAXE n'est pas en mesure de traiter des vitesses trop élevées. Il est recommandé de limiter la vitesse de transfert de l'émetteur en positionnant une attente (par exemple 2 ms) entre chaque octet. . Un maximum de 4800 bauds est conseillé pour les transactions complexes.

Cette commande bloque le programme en attente d'un octet (ou plus) sur l'entrée activée, elle ne peut pas être interrompue par une commande SETINT.

Après utilisation de cette commande (en cas de blocage dans la procédure serin), il est possible d'effectuer un « reset manuel » pour télécharger un nouveau programme. Les PICAXEs équipés de résonateurs ou quartz externes sont plus précis que ceux utilisant des résonateurs internes. Il est possible de calibrer le PICAXE avec la commande calibfreq.

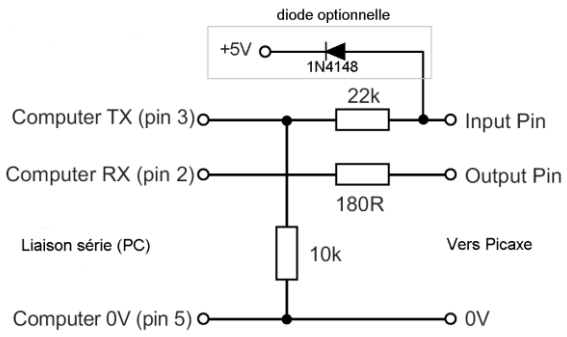
Exemple:

```
      ; le premier octet reçu est positionné dans b1
      ; Attention, la commande serin bloque le programme en attente d'une donnée
      serin 1,N2400,b1   ; réception de la donnée sur l'entrée 1, mode N 2400 bauds

      ; la donnée série est positionnée dans b1 si la chaîne "ABC" est reçue
      ; Attention, la commande serin bloque le programme en attente des données
      serin 1,N2400,("ABC"),b1

      ; attente de la séquence "go" avant poursuite du programme
      serin 1,N2400,("go")

      ; uniquement sur les séries M2, X1 et X2
      ; si l'entrée 1 ne reçoit pas de données pendant plus de 100ms, le programme
continue
      serin [100], 1, N2400, b1

      ; uniquement sur les séries M2, X1 et X2
; si l'entrée 1 ne reçoit pas de données pendant plus de 100ms, le programme se déroute
vers la procédure noreception
      serin [100, noreception], 1, N2400, b1
```

Exemple de circuit d'interface simple :

Pour les circuits 8 ou 14 broches, la structure interne de l'entrée 3 (C.3) nécessite la présence d'une diode 1N4148.

Pour les circuits 20 broches, la structure interne de l'entrée 6 (C.6) nécessite la présence d'une diode 1N4148.



(c) Revolution Education Ltd.
All rights reserved

2.133 serrxd

207

Syntaxe:

SERRXD (qualifier, qualifier…)

SERRXD (qualifier, qualifier…), {#}variable, {#}variable…

SERRXD {#}variable, {#}variable…

Options de syntaxe indiquant un délai optionnel pour les séries M2, X1 et X2:

SERRXD [timeout], (qualifier…)

SERRXD [timeout], (qualifier…), {#}variable, {#}variable

SERRXD [timeout], {#}variable, {#}variable

SERRXD [timeout, address], (qualifier…)

SERRXD [timeout, address], (qualifier…), {#}variable, {#}variable…

SERRXD [timeout, address], {#}variable, {#}variable…

- qualifier est une variable/constante (de 0 à 255) qui doit être reçu avant la prise en compte des octets suivants
- variable est la variable dans laquelle on positionne la donnée reçue (de 0 à 127)
- timeout permet de choisir le temps d'attente d'une réception (en milliseconde)
- address correspond au nom de label vers lequel le programme est dérouté si le temps d'attente est dépassé

Fonction:

Permet de recevoir une donnée en format série sur la broche de programmation du PICAXE à la vitesse de transmission fixe de 4800 bauds (9600 bauds sur les séries X2) en mode N.

Information:

La commande serrxd est très proche de la commande serin, cependant elle utilise le jack de programmation via le câble de programmation AXE027 (série ou USB).

Le PICAXE vérifie en permanence la broche de téléchargement pour vérifier si un ordinateur tente d'initialiser un nouveau téléchargement, c'est pourquoi la commande serrxd pourra être précédé de la commande disconnect.

Après utilisation de cette commande, il est toujours possible d'effectuer un « reset manuel » pour télécharger un nouveau programme.

Modification de la vitesse d'horloge :

Toute modification de la fréquence d'horloge (utilisation de setfreq par exemple) modifiera la vitesse de transmission :

4MHz	8MHz	16MHz	32MHz
4800 bauds	9600 bauds	19200 bauds	38400 bauds

Exemple:

```
; la donnée série est positionnée dans b1 si la chaîne "ABC" est reçue
; Attention, la commande serrxd bloque le programme en attente des données
Serrxd ("ABC"),b1

; disconnect puis reconnect du PICAXE lors de l'utilisation de serrxd
disconnect
Serrxd [1000,timeout],@ptrinc,@ptrinc,@ptr
reconnect
```

2.134 serout

208

Syntaxe:

SEROUT pin, baudmode, {#}data, {#}data…

- pin indique le numéro de la broche de sortie utilisée par le PICAXE.
- baudmode permet de choisir le mode (N ou T) et la vitesse de fonctionnement

Txxx indique le mode normal Nxxx indique le mode inversé

Vitesse de transmission possible pour la série 08 / 08M / 18 / 18A / 28 / 28A :

4MHz	8MHz	16MHz
T300_4	T600_8	T1200_16
T600_4	T1200_8	T2400_16
T1200_4	T2400_8	T4800_16
T2400_4	T4800_8	T9600_16
N300_4	N600_8	N1200_16
N600_4	N1200_8	N2400_16
N1200_4	N2400_8	N4800_16
N2400_4	N4800_8	N9600_16

Vitesse de transmission possible pour les autres séries :

4MHz	8MHz	16MHz	32 MHz	64MHz
T600_4	T1200_8	T2400_16	T4800_32	T9600_64
T1200_4	T2400_8	T4800_16	T9600_32	T19200_64
T2400_4	T4800_8	T9600_16	T19200_32	T38400_64
T4800_4	T9600_8	T19200_16	T38400_32	T76800_64
N600_4	N1200_8	N2400_16	N4800_32	N9600_64
N1200_4	N2400_8	N4800_16	N9600_32	N19200_64
N2400_4	N4800_8	N9600_16	N19200_32	N38400_64
N4800_4	N9600_8	N19200_16	N38400_32	N76800_64

- data est la donnée que l'on veut émettre (de 0 à 255), variable possible.

Fonction:

Permet d'émettre une ou plusieurs données en format série sur une broche de sortie du PICAXE (pas de parité, 8 bits de données, 1 bit d'arrêt).

Information:

Cette commande ne doit pas utiliser la broche de téléchargement qui demande l'utilisation de la commande sertxd.

Si vous utilisez une interface simple à résistances, il faut choisir le mode N (mode inversé).

Si vous utilisez une interface avec un circuit MAX232, il faut choisir le mode T.

Si vous utilisez le mode T, le premier octet peut être erroné si la broche de sortie est au niveau bas avant la commande serout. Pour éviter ce problème, il est recommandé de positionner la broche de sortie au niveau haut avant l'utilisation de la première commande serout.

Un texte doit être entre guillemets : ″Bonjour″,

Le symbole # permet de fournir une donnée en format ASCII. Par conséquent si b1 = 126, #b1 va afficher les caractères ″1″,″2″,″6″ plutôt que la donnée 126.

Les PICAXEs équipés de résonateurs ou quartz externes sont plus précis que ceux utilisant des résonateurs internes. Il est possible de calibrer le PICAXE avec la commande calibfreq.

(c) Revolution Education Ltd.
All rights reserved

Exemple:

```
debut:
        for b0 = 0 to 20                    ; début de la boucle pour compter de 0 à 20
            serout7,N2400,(254,128,#b0)     ; transmission de la valeur b0 à l'afficheur LCD
        next b0
        end
```


(c) Revolution Education Ltd.
All rights reserved

2.135 sertxd

210

Syntaxe:

SERTXD ({#}data, {#}data…)

- data est la donnée que l'on veut émettre (de 0 à 255), variable possible.

Fonction:

Permet d'émettre une ou plusieurs données en format série sur la broche de programmation du PICAXE à la vitesse de transmission fixe de 4800 bauds (9600 bauds sur les séries X2) en mode N.

Information:

La commande sertxd est très proche de la commande serout, cependant elle utilise le jack de programmation via le câble de programmation AXE027 (série ou USB).

Cette commande permet d'effectuer, par exemple, le débogage d'un programme en visualisant les données souhaitées dans une fenêtre Terminal (utiliser l'option PICAXE > Terminal de Programming Editor).

Modification de la vitesse d'horloge :

Toute modification de la fréquence d'horloge (utilisation de setfreq par exemple) modifiera la vitesse de transmission :

4MHz	8MHz	16MHz	32MHz
4800 bauds	9600 bauds	19200 bauds	38400 bauds

Exemple:

```
debut:
      for b0 = 0 to 20                    ; début de la boucle pour compter de 0 à 20
          sertxd ("valeur de b0: ",#b0,13,10) ; transmission de la valeur b0
                                              ; 13 permet un retour chariot et 10 un saut de ligne
          pause 1000
      next b0
      end
```


(c) Revolution Education Ltd.
All rights reserved

2.136 servo

211

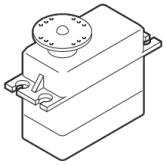
(tous les PICAXEs)

Syntaxe:

SERVO broche,Impulsion

SERVO [preload],broche,Impulsion (X2 only)

- Broche est une variable/constante spécifiant l'entrée/sortie broche utilisée.
- Impulsion est une variable/constante (75-225) spécifiant la position du servo
- Preload est une constante de temps optionnelle (X2 seulement).



Fonction:

Génère en continu une impulsion sur une broche de sortie pour commander un servo de type radio

Sur les M2 et X2 la commande servo ne fonctionne que sur le portB (B.0 à B.7)

Information:

Les servos, comme ceux que l'on trouve couramment dans les jouets radio commandés,sont des moto-réducteurs très précis qui peuvent conserver leur position grâce à leur capteur de position interne.

Généralement les servos nécessitent des impulsions de 0.75 to 2.25ms toutes les 20ms, et ces impulsions doivent être répétées constamment toutes les 20 ms.

Une fois que l'impulsion est perdue, le servo perd sa position

La commande servo envoie une impulsion à l'état haut sur une broche , pendant une durée de la valeur de impulsion * 0.01ms toutes les 20 ms

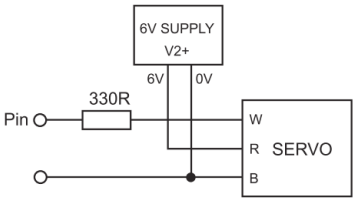
Cette commande est différente de celle rencontrée sur la plupart des autres Basic en ce sens qu'elle est continue à moins qu'une autre commande servo, ou une instruction High ou Low soit exécutée.

Les commandes High et low stoppent les impulsions immédiatement.

La commande servo ajuste la longueur d'impulsion à la nouvelle valeur, engendrant le déplacement du servo.

Servo ne peut être utilisé en même temps que timer ou pwmout/hpwm car ils utilisent le même timer interne.

La commande servo initialise la broche pour la fonction et démarre le timer. Une fois que la broche a été initialisée, il est recommandé de d'utiliser la commande servopos pour changer la position ? Cela évite de réinitialiser le timer, ce qui peut conduire à des vibrations.



Eviter d'utiliser des valeurs d'impulsion inférieure à 75, ou supérieure à 225, car cela peut entraîner des dysfonctionnements.

Compte tenu des tolérances de fabrication des servos, ces valeurs sont approximatives et nécessitent des ajustements par expérimentation. (par ex. 60 à 200).

Toujours utiliser une alimentation séparée de 6V (par ex. 4 piles AA) pour les servos car ils génèrent beaucoup de parasites

Noter que le temps de traitement nécessaire pour activer la commande servo peut entraîner un retard sur d'autres commandes, telle la pause qui peut être un peu plus longue que prévu.

Les impulsions servo sont également désactivées durant des commandes liées au temps, telles que serin, serout, sertxd, debug etc.

Sur les PICAXEs X2 servo ne fonctionnera qu'à 8MHz ou 32MHz.

(c) Revolution Education Ltd.
All rights reserved

Sur les PICAXEs M2 et X1 servo ne fonctionnera qu'à 4MHz or 16MHz.

Sur tous les autres, servo fonctionnera à 4MHz.

Sur les PICAXEs X2 il est possible de changer les 20 ms de délai entre impulsions. Cela est réalisable par la valeur 'preload' qui est le nombre à précharger dans le timer 1 avant le début du comptage. Sur les X2 , timer 1 s'incrémenté toutes les 0.5µs, donc pour un délai de 20ms (20,000µs) nous avons besoin de 40 000 incréments.

Aussi, la valeur de preload est de 65536 - 40000 = 25536.

Par exemple, pour des servos numériques, vous pouvez souhaiter augmenter la fréquence des impulsions tous les 10 ms

(notez que le délai doit être plus long que la somme de toutes les impulsions de l'ensemble des servos. donc 10ms est uniquement possible pour un maxi de 4 servos (le délai maximum pour 4 servos est avec l'impulsion de 2.25ms soit, 4x2.25 = 9ms).

10ms = 10000 µs = 20 000 pas

65536-20000 = 45536

Donc la commande sera :
```
servo [45536],1,75
```

Effet de l'accroissement de la fréquence d'horloge:

La commande servo fonctionnera correctement à 4MHz sur tous les PICAXEs (exceptés les X2, qui ne fonctionnent qu'à 8 ou 32MHz). 16MHz est supporté sur les M2 and X1. Aucune autre fréquence ne fonctionnera correctement.

Exemple:
```
init: servo 4,75          ; initialise le servo
main: servopos 4,75       ; déplacement du servo en fin de course
pause 2000                ; attendre 2 secondes
servopos 4,225            ; déplacement du servo vers l'autre fin de course
pause 2000                ; attendre 2 secondes
goto main                 ; reboucle au début.
```

2.137 servopos

213

(tous les PICAXEs)

Syntaxe:

SERVOPOS broche,Impulsion

SERVOPOS broche,OFF

- Broche est une variable/constante spécifiant l'entrée/sortie broche utilisée.
- Impulsion est une variable/constante (75-225) spécifiant la position du servo

Fonction:

Ajuste la longueur d'impulsion appliquée à un servo de type radio commande pour modifier sa position. Une commande servo sur la même broche doit avoir été préalablement envoyée.

Information:

Les servos, comme ceux que l'on trouve couramment dans les jouets radio commandés,sont des moto-réducteurs très précis qui peuvent conserver leur position grâce à leur capteur de position interne.

Généralement les servos nécessitent des impulsions de 0.75 to 2.25ms toutes les 20ms, et ces impulsions doivent être répétées constamment toutes les 20 ms.

Une fois que l'impulsion est perdue, le servo perd sa position.

La commande servo envoie une impulsion à l'état haut sur une broche , pendant une durée de la valeur de impulsion * 0.01ms toutes les 20 ms.

La commande servo initialise la broche pour la fonction et démarre le timer. Une fois que la broche a été initialisée, il est recommandé d'utiliser la commande servopos pour changer la position. Cela évite de réinitialiser le timer, ce qui peut conduire à des vibrations.

Eviter d'utiliser des valeurs d'impulsion inférieure à 75, ou supérieure à 225, car cela peut entraîner des dysfonctionnements.

Compte tenu des tolérances de fabrication des servos, ces valeurs sont approximatives et nécessitent des ajustements par expérimentation. (par ex. 60 à 200).

Toujours utiliser une alimentation séparée de 6V (par ex. 4 piles AA) pour les servos car ils génèrent beaucoup de parasites.

Noter que le temps de traitement nécessaire pour activer la commande servo peut entraîner un retard sur d'autres commandes, telle la pause qui peut être un peu plus longue que prévue.

Les impulsions servo sont également désactivées durant des commandes liées au temps, telles que serin, serout, sertxd, debug etc.

Sur les PICAXEs X2 servo ne fonctionnera qu'à 8MHz ou 32MHz.

Sur les PICAXEs M2 et X1 servo ne fonctionnera qu'à 4MHz or 16MHz.

Sur tous les autres, servo fonctionnera à 4MHz.

Effet de l'accroissement de la fréquence d'horloge:

La commande servo fonctionnera correctement à 4MHz sur tous les PICAXEs (exceptés les X2, qui ne fonctionnent qu'à 8 ou 32MHz). 16MHz est supporté sur les M2 and X1. Aucune autre fréquence ne fonctionnera correctement.

Exemple:

```
init: servo 4,75          ; initialise le servo
main: servopos 4,75       ; déplacement du servo en fin de course
pause 2000                ; attendre 2 secondes
servopos 4,225            ; déplacement du servo vers l'autre fin de course
pause 2000                ; attendre 2 secondes
goto main                 ; reboucle au début
```

2.138 setbit

214

Syntaxe:

SETBIT var, bit

"var" est la variable cible
"bit" est le bit à forcer. dans cette variable (0-7 pour un octet, 0-15 pour un mot)

Concerne :

20X2

28X1 / 28X2

40X1 / 40X2

Commentaire :

Force en 1 le bit spécifié dans la variable spécifiée

Exemples :

```
setbit b6, 0        ;force en 1 le bit 0 de l' octet b6
setbit w4, 15       ; force en 1 le bit 15 du mot w4
```


(c) Revolution Education Ltd.
All rights reserved

2.139 setint

215

(tous les PICAXEs)

Syntaxe:

SETINT OFF

SETINT entrée,mask (condition ET)

SETINT AND entrée,masque (condition ET)

Options pour les M2, X1 et X2:

SETINT OR entrée,masque (Condition OU)

SETINT NOT entrée,masque (Condition NON ET)

Options pour les X2:

SETINT entrée,masque,port

SETINT NOT entrée,masque,port
- entrée est une variable/constante (0-255) spécifiant la condition d'entrée.
- masque est une variable/constante (0-255) qui spécifie the masque.
- port est le port sur les X2 (A,B,C,D) *voir les restrictions ci après !*

Fonction:

Crée une interruption sur certaines conditions d'entrées.

Les X1 et X2 peuvent également générer une interruption sur certains byte drapeau ('flags') -

Voir la commande **setintflags**.

Information:

La commande setint génère une interruption en fonction de certaines conditions d'entrée.

Cela peut être une combinaison de broches sur le port d'entrée (portC).

Les X2 peuvent également vérifier un port différent si nécessaire.

Par défaut la condition logique est un ET (AND) entre les broches sélectionnées.

Sur certains PICAXEs, il est possible d'utiliser la négation de cette condition ET (NON ET ou NAND).

Sur certains PICAXEs, il est possible d'utiliser la condition logique OU (OR) entre les broches sélectionnées .

Une interruption est un moyen rapide de réagir en fonction d'une combinaison d'entrées particulière.

C'est le seul type d'interruption possible avec les PICAXEs. Le port d'entrées est lu entre chaque exécution de chaque ligne de commande du programme, entre chaque note d'une commande son, et continuellement pendant une commande pause.

Si la condition particulière d'entrées est vraie, un 'gosub' vers le sous programme d'interruption est immédiatement exécuté.

Quand le sous programme est effectué, l'exécution du programme principal continue.

La condition d'interruption sur les entrées est une combinaison de '0's et de '1's sur le port d'entrée, masqué par le byte 'masque'. Aussi, tout bit masque par un 0 du byte masque sera ignoré.

Pour une interruption seulement sur l'entrée 1 à l'état haut :

```
setint %00000010,%00000010
```

Pour une interruption seulement sur l'entrée 1 à l'état bas :

```
setint %00000000,%00000010
```

Pour une interruption sur l'entrée 0 à l'état haut, l'entrée 1 à l'état haut et l'entrée 2 à l'état bas :

```
setint %00000011,%00000111
```

etc.

Une seule combinaison d'entrées à la fois est autorisée. Pour désactiver l'interruption, exécuter la commande **SETINT OFF**.

Les M2, X1, X2 supportent la condition NON (NOT) où l'interruption intervient quand la combinaison n'est pas le masque défini.

Ils peuvent également utiliser le byte 'flags' (au lieu du port d'entrée) pour générer la condition d'interruption.

Restrictions.

Compte tenu de la configuration interne de certains Pic, une limite d'utilisation de certaines broches existe:

Le port d'entrée par défaut est portC. (c'est-à-dire s'il n'est pas explicitement déclaré dans **port** de la commande)

14M/14M2 : seules les entrées 0,1,2 peuvent être utilisées

20M seules les entrées 1-5 peuvent être utilisées

20M2/20X2 seul le portC peut être utilisé, et seulement C.1 à C.5 du portC (ne concerne que le 20M2 et le 20X2, pas le 28X2)

40X2 sur le portA, seules A.0 to A.3 peuvent être utilisées

Notes:

1) Chaque programme utilisant la commande SETINT doit avoir à la fin, un sous programme interrupt: se terminant par une commande return.
2) Quand l'interruption survient, l'interruption est désactivée; Aussi pour réactiver l'interruption (si nécessaire) une commande SETINT doit être utilisée à l'intérieur même du sous programme interrupt:
3) L'interruption ne sera pas active tout pendant que la commande return n'est pas effectuée.
4) Si l'interruption est réactive alors que la condition d'interruption n'est pas inhibée à l'intérieur du sous programme, une seconde interruption peut survenir dès la commande return.
5) Après l'exécution du code d'interruption, le programme continue à la ligne suivante du programme principal.

Dans le cas d'une interruption intervenant lors d'une commande pause, play, ou tune le temps restant est ignoré et le programme continue à la ligne suivante.

Explication plus détaillées de SETINT.

SETINT doit être suivi de deux nombres : entrées à comparer et un masque , dans cet ordre.

Il est normal de faire figurer ces nombres dans un format binaire, rendant la lecture des entrées actives plus lisible.

Dans le format binaire, l'entrée7 est à gauche et l'entrée 0 est à droite.

Le second nombre, l'entrée masque, définit quelles broches doivent être verifiées pour savoir si une interruption doit être générée ...
- %00000001 vérifie l'entrée broche 0
- %00000010 vérifie l'entrée broche 1
- %01000000 vérifie l'entrée broche 6
- %10000000 vérifie l'entrée broche 7
- etc

On peut combiner cela avec le contrôle de plusieurs entrée à la fois.
- %00000011 vérifie l'entrée pins 1 and 0
- %10000100 vérifie l'entrée pins 7 and 2

Après avoir décidé quelles broches doivent être utilisées pour l'interruption, le premier nombre (valeur d'entrées) définit l'état dans lequel elles doivent être ,on (1) ou off (0).

Une fois que setint est actif, le PICAXE ne surveille que les broches spécifiées dans la valeur "masque", par un 1 présent, et ignore les autres. Une entrée masque de %10000100 contrôle les boches 7 et 2 et crée une valeur %a0000b00 où le bit 'a' sera 1 si la broche 7 est à létat haut et 0 si elle est à l'état bas et le bit 'b' sera 1 si la broche 2 est à létat haut et 0 si elle est à l'état bas

Si l'entrée masque est %10000100, pour les broches 7 et 2, alors la comparaison valide peut être l'une des suivantes :
- %00000000 Broche 7 = 0 et broche 2 = 0
- %00000100 Broche 7 = 0 et broche 2 = 1
- %10000000 Broche 7 = 1 et broche 2 = 0
- %10000100 Broche 7 = 1 et broche 2 = 1

Aussi si vous voulez générer une interruption si la Broche 7 est à l'état haut et la Broche 2 est à l'état bas, l'entrée masque est %10000100 la valeur de comparaison est %10000000, donnant pour instruction ;
- **SETINT %10000000,%10000100**

L'interruption interviendra exclusivement quand la broche 7 est à l'état haut et la broche 2 est à l'état bas.

Exemple:

```
setint %10000000,%10000000 ; active l'interruption quand la broche 7 seule passe à l'état haut
main:
        low 1                       ; met la sortie 1 à l'état bas
        pause 2000                  ; attend 2 secondes
        goto main                   ; boucle en début de programme

        interrupt:
        high 1                      ; met la sortie 1 à l'état haut
        if pin7 = 1 then interrupt  ; boucle jusqu'à  la fin de la condition
        pause 2000                  ; attend 2 secondes
        setint %10000000,%10000000  ; re-active l' interruption
        return                      ; retour du sous programme
```

Dans cet exemple une LED sur la sortie 1 s'allumera immédiatement dès que l'interrupteur est actionné. Avec un classique **if pin7 =1 then....** le programme pourrait demander jusqu'à 2 secondes pour allumer la LED car le **If** n'est pas activé durant la commande **pause 200** dans la boucle du programme principal. (voir le programme classique ci-dessous, pour comparer.)

```
main:
        low 1             ; met la sortie 1 à l'état bas
        pause 2000        ; attend 2 secondes
        if pin7 = 1 then sw_on
        goto main         ; retour au programme principal

sw_on:
        high 1                      ; met la sortie 1 à l'état haut
        if pin7 = 1 then sw_on      ; boucle jusqu'à ce que la condition soit fausse
        pause 2000                  ; attend 2 secondes
        goto main                   ; retour au programme principal
```


(c) Revolution Education Ltd.
All rights reserved

2.140 setintflags

219

(PICAXEs 20X2, 28X1, 28X2, 40X1, 40X2)

Syntaxe:

SETINTFLAGS OFF

SETINTFLAGS flags,masque

SETINTFLAGS AND flags,masque

SETINTFLAGS OR flags,masque

SETINTFLAGS NOT flags,masque

- flags est une variable/constante (0-255) définissant le byte d'état des conditions.
- masque est une variable/constante (0-255) définissant le masque

Fonction:

Génère une interruption sur certaines conditions du byte .
Voir l'usage en détail dans la commande **'setint'** command, qui s'applique également à la commande **setintflags**.
Une seule interruption peut être active à la fois.

Information:

La commande setintflags génère une interruption en fonction de certaines conditions des flags.
Une interruption est un moyen rapide de réagir en fonction d'une combinaison d'entrées particulière.
C'est le seul type d'interruption possible avec les PICAXEs. Le byte de flags est lu entre chaque exécution de chaque ligne de commande du programme, entre chaque note d'une commande son, et continuellement pendant une commande pause.
Si la condition particulière d'entrées est vraie, un 'gosub' vers le sous programme d'interruption est immédiatement exécuté.
Quand le sous programme est effectué, l'exécution du programme principal continue.
La condition d'interruption sur les entrées est une combinaison de '0's et de '1's du byte de flags, masqué par le byte 'masque'.
Aussi, tout bit masque par un 0 du byte masque sera ignoré.
Le byte de flags est découpé en variables individuelle de bits.
Voir les commandes détaillées pour les détails spécifiques à chaque flag :

Nom		fonction spéciale	Command
flag0	hint0flag	sur X2 - interruption INT0	hintsetup
flag1	hint1flag	sur X2 parts - interruption INT1	hintsetup
flag2	hint2flag	sur X2 parts - interruption INT2	hintsetup
flag3	hintflag	sur X2 parts - interruption sur broches 0,1,2	hintsetup
flag4	compflag	sur X2 parts - flag du comparateur	compsetup
flag5	hserflag	si réception en arrière plan sur hserial	hsersetup
flag6	hi2cflag	si écriture hi2c (mode esclave)	hi2csetup
flag7	toflag	flag de dépassement de capacité du timer	settimer

pour une interruption sur le dépassement de capacité du timer 0

```
setintflags %10000000,%10000000
```

pour une interruption lors d'une écriture hi2c (mode esclave)

```
setintflags %01000000,%01000000
```

pour une interruption lors d'une reception série en arrière plan

```
setintflags %00100000,%00100000
```

Une seule combinaison à la fois, est possible. Pour desactiver l'interruption, executer la commande **setintflags off** .
Pour plus d'informations sur les options de setintflags (AND / OR / NOT), voir la commande **setint**

Exemple:

```
setintflags %10000000,%10000000 ;configure une interruption avec le timer0
```


(c) Revolution Education Ltd.
All rights reserved

2.141 setfreq

221

(Tous les PICAXEs)

Syntaxe:

setfreq freq

- freq est le mot clé définissant la fréquence

08M, 14M, 20M	interne	m4, m8
18A, 18M, 18X	interne	m4, m8
All M2 parts	interne	k31, k250, k500, m1, m2, m4, m8,m16,m32
20X2	interne	k31, k250, k500, m1, m2, m4, m8, m16, m32 ,m64
28X1, 40X1	interne	k31,k125,k250,k500,m1, m2, m4, m8
	externe	em4, em8, em10, em16, em20
28X2, 40X2	interne	k31, k250, k500,m1, m2, m4, m8, m16
	externe	em16, em32, em40, em64
28X2-5V, 40X2-5V	interne	k31, k250, k500,m1, m2, m4, m8
	externe	em16, em32,em40
28X2-3V, 40X2-3V	interne	k31, k250, k500,m1, m2, m4, m8, m16
	externe	em16, em32, em40, em64

où k31 = résonateur interne 31kHz
m4 = résonateur interne 4MHz
em16 = résonateur externe 16MHz

Fonction:

Définit la fréquence interne du microcontrôleur avec le résonnateur interne à 8MHz (m8) ou d'autres valeurs.

La valeur par défaut sur les X2 est 8MHz interne. La valeur par défaut sur tous les autres est 4MHz interne.

Information:

La commande **setfreq** peut être utilisée pour changer la vitesse de traitement du microcontrôleur de 4 à 8MHz ou autres valeurs.

Toutefois, il faut noter que cette vitesse affecte de nombreuses commandes en changeant par exemple leur durée (par exemple une pause durera deux fois moins longtemps à 8MHz).

Noter que les X2 ont un PLL x4 en interne. Cela multiplie la fréquence du résonnateur externe par 4.

Aussi la valeur du résonnateur externe à utiliser doit être le 1/4 de la fréquence finale souhaitée.

(par ex. Le mode **em40** utilise un résonnateur externe de 10MHz, et un résonnateur 4MHz pour **em16**).

Le changement est immédiat. Si une commande **setfreq** n'est pas utilisée, tous les programmes seront à (4MHz) (ou m8, 8MHz sur les X2).

Noter que Programming Editor supporte seulement certaines fréquences pour le téléchargement. Si le PICAXE s'exécute à des fréquences différentes les M2, X1 and X2 commuteront automatiquement sur le résonnateur interne par défaut, 4MHz ou 8MHz lors du téléchargement.

Sur les M2 , avec les programmes multi tâches la commande **setfreq** ne peut pas être utilisée car la fréquence de l'oscillateur est alors sous contrôle du firmware PICAXE.

La fréquence du résonnateur interne est étalonnée avec la meilleure précision en usine.

Toutefois les utilisateurs avertis peuvent utilizer la commande calibfreq pour ajuster ces valeurs préétablies.

Certaines commandes telles **readtemp** ne fonctionnent qu'à 4MHz. Dans ce cas, revenir temporairement à 4MHz. (sur les M2, X1 et X2 cela est automatique).

Noter que ce changement temporaire (automatique ou non) aura un effet sur les tâche d'arrière plan telles que pwmout / hpwm.

Exemple:

```
setfreq em32              ; frequence externe à 32MHz
pause 4000                ; ne sera pas 4 secondes
setfreq m4                ; setfreq à 4MHz
readtemp 1,b1     ; execute la commande à 4MHz
setfreq em32              ; revient à 32MHz
```


(c) Revolution Education Ltd.
All rights reserved

2.142 settimer

223

(PICAXEs 20X2 28X1 28X2 40X1 40X2)

Syntaxe:

SETTIMER OFF

SETTIMER preload

SETTIMER COUNT preload

- preload est une constante/variable qui définit la période du timer.

  Par facilité, la valeur 1s est prédéfinie dans le compilateur
  t1s_4 (valeur de préchargement 49910 - 1 seconde à 4MHz)
  t1s_8 (valeur de préchargement 34286 - 1 seconde à 8MHz)
  t1s_16 (valeur de préchargement 3036 - 1 seconde à 16MHz)

Fonction:
Configure et active l'horloge / compteur interne.

Information:

La commande settimer est utilisée pour configurer la fonction horloge / compteur (timer)

La fonction **timer** peut être utilisée de deux façons – comme une horloge interne ou comme compteur externe (entrée 0 (C.0) seulement).

Noter que la commande **debug** désactive le timer temporairement (durant la transmission de variables)

L'utilisation de **debug** en même temps conduira à des erreurs de lecture.

Compteur Externe (non disponible sur 20X2)

Dans le mode compteur externe un registre compteur interne (non accessible à l'utilisateur final) est incrémenté à chaque front montant détecté sur l'entrée 0. Ce comptage d'impulsion se fait en arrière plan, aussi le programme PICAXE peut exécuter d'autres tâches en même temps (à la différence de la commande count qui arrête les autres process durant la période de comptage).

Quand le registre de comptage interne passe de 65535 à 0 (dépassement de capacité) la variable spéciale timer est automatiquement incrémentée.

Donc pour incrémenter la variable timer toutes les 10 impulsions externes, la valeur preload devra être 65535 – 10 =65526.

Après 10 impulsions le registre de comptage sera en dépassement et incrémentera la variable timer. Pour incrémenter timer à chaque impulsion externe, mettre la valeur 65535 en preload.

Si la variable timer passe en dépassement de capacité (c.a.d, si elle passe de 65535 à 0) le flag **toflag** (timer overflow flag) est mis à 1. Le flag **toflag** est remis à 0 avec la commande settimer mais peut aussi être remis à 0 manuellement dans le programme par un **let toflag = 0**. Si souhaité, une interruption peut être programmée pour détecter ce flag. Voir la commande **setintflags**.

Exemple:

```
settimer count 65535 ' settimer en mode count
main:
pause 10000      ' attend 10 secondes, de comptage d'impulsions
debug            ' affiche la valeur de timer
goto main        ' loop
```

Timer Interne

Dans le mode timer interne le temps passé est stocké dans la variable timer de type word à laquelle on peut accéder comme n'importe quelle variable. Par exemple: **if timer > 200 then skip**

Quand la variable timer passe de 65535 à 0 le flag (toflag) mis à 1 . Le flag toflag est remis à 0 avec la commande settimer mais peut aussi être remis à 0 manuellement dans le programme par un let toflag = 0.

Si souhaité, une interruption peut être programmée pour détecter ce flag. Voir la commande setintflags.

La période du timer peut être définie par l'utilisateur: Le timer fonctionne avec des impulsions primaires et des impulsions secondaires. Une impulsions primaire survient toute les 256/(fréquence horloge) secondes.

Avec un résonateur à 4MHz cela signifie que cette impulsion est générée toutes les 64µs (32µs à 8MHz, 16µs à 16MHz, 8µs à 32MHz, 4µs à 64MHz). Quand la variable word (non accessible) liée à ces impulsions primaires passe de 65535 to 0, une impulsion secondaire est générée.

Elle incrémente la variable timer, et le nombre d'impulsions est connu par la lecture de la variable timer.

La valeur preload est utilisée pour définir le nombre d'impulsions nécessaires avant d'être en dépassement de capacité (65535).

Cela signifie qu'il n'est pas nécessaire d'attendre 65536 impulsions pour incrémenter le compteur secondaire.

Par exemple si la valeur preload est 60000 il faudra 5536 impulsions primaires pour générer un secondaire.

Par exemple supposons que vous souhaitiez que le compteur s'incrémente chaque seconds à 4MHz.

On sait qu'à 4MHz chaque 64us est générée une impulsion primaire. Une seconde est égale à 1000000 µs.

Donc on a besoin de 1000000 / 64 soit 15625 impulsions pour une période de 1 seconde.

La valeur de preload sera donc 65536 - 15625 = 49910.

Timer ne peut pas être utilisé en même temps que la commande servo, car la commande servo nécessite l'usage exclusif du timer pour calculer les intervalles entre impulsions du servo.

Exemple:

```
settimer t1s_4          ' settimer avec 1 seconde d'intervalle à 4MHz
                        ' identique à settimer 49910 (NdT)
main:
pause 10000             ' attend 10 secondes
debug                   ' affiche la valeur timer
goto main               ' retour de boucle
```

2.143 shiftin (spiin)

225

(Tous les PICAXEs)

Syntaxe:

SPIIN sclk,sdata,mode,(variable {/ bits} {, variable {/ bits}, ...})

- sclk est une variable/constante qui spécifie la broche entrée/sortie utilisée en horloge (clock).
- sdata est une variable/constante qui spécifie la broche entrée/sortie utilisée en données (data).
- Mode est une variable/constante (0-7) qui spécifie le mode:

0	MSBPre_L	(MSB first, sample before clock, idles low)
1	LSBPre_L	(LSB first, sample before clock, idles low)
2	MSBPost_L	(MSB first, sample after clock, idles low)
3	LSBPost_L	(LSB first, sample after clock, idles low)
4	MSBPre _H	(MSB first, sample before clock, idles high)
5	LSBPre_H	(LSB first, sample before clock, idles high)
6	MSBPost_H	(MSB first, sample after clock, idles high)
7	LSBPost _H	(LSB first, sample after clock, idles high)

- Variable reçoit les données.
- Bits est le nombre optionnel de bits à transmettre. S'il est omis, le nombre par défaut est 8.

Information:

La commande spiin (shiftin est également accepté par le compilateur) est une méthode 'bit-bang' method de communication SPI sur les X1 et X2 uniquement. Tous les autres PICAXEs doivent utiliser le programme exemple inclus pour bénéficier de la même fonctionnalité.
Pour une solution hardware avec les X1/X2 parts voir la commande **hshin**.
Par défaut, 8 bits sont décalés dans la variable. Un nombre de bits différent (1 to 8) peut être définie par l'option / bits.
Par exemple, si l'on souhaite décaler sur 12 bits, le faire sur deux bytes, un byte décalant 8 bits et le second byte décalant 4 bits.
Noter que si on utilise la méthode LSB first les bits sont décalés vers la droite, et ainsi, le décalage de 4 bits laissera ceux occupant les bits7-4 (et non 3-0). Avec la méthode MSB les bits sont décalés vers la gauche.
Lors d'une connexion d'un périphérique SPI (par ex. EEPROM) se rappeler que le data-in de l'EEPROM se connecte à data-out du PICAXE, et vice et versa.
Les autres PICAXEs n'ont pas de commande directe spiin (shiftin).
Toutefois la même fonctionnalité peut être trouvée en utilisant la procédure décrite ci-dessous.

Effets de l'augmentation de la fréquence d'horloge.

Augmenter le fréquence d'horloge augmente la fréquence d'horloge du SPI .

Exemple:

```
spiin 2,1,LSB_Pre_H, (b1 / 8) ' clock 8 bits into b1
```

shiftin/shiftout sur les PICAXEs avec des commandes natives:

Certains PICAXE n'ont pas de commande shiftin. Toutefois la même fonctionnalité peut être trouvée en utilisant la procédure décrite ci-dessous.
Ces procédures sont aussi places dans le fichier nommé shiftin_out.bas dans le dossier **\samples** du logiciel Programming Editor.
Pour les utiliser, copier simplement les définitions de symboles en tête de votre programme et copier la procédure shiftin en bas du programme.
Ne pas copier toutes les options pour économiser de la mémoire.
Il est sous entendu que les sorties data et clock (sdata and sclk) sont à l'état bas avant d'utiliser le gosub.
La ligne BASIC

"shiftin sclk, sdata,mode, (var_in(\bits)) "

devient

```
gosub shiftin_LSB_Pre     (for mode LSBPre)
gosub shiftin_MSB_Pre     (for mode MSBPre)
gosub shiftin_LSB_Post    (for mode LSBPost)
gosub shiftin_MSB_Post    (for mode MSBPost) '
```

```
' ~~~~~ SYMBOL DEFINITIONS ~~~~~
```


(c) Revolution Education Ltd.
All rights reserved

```
' Nécessaire pour toutes les routines. Changer les numéros de broches et bits comme indiqué .
' Utilise les variables b7-b13 (c.a.d. b7,w4,w5,w6). Si seuls 8 bits sont utilisés
' toutes les variables word peuvent être changées en byte.
'
'***** Exemple de définitions de symboles *****
symbol sclk = 5           ' broche horloge (sortie)
symbol sdata = 7          ' données (broche sortie pour shiftout)
symbol serdata = entrée7  ' données (broche entrée for shiftin, note entrée7
symbol counter = b7       ' variable utilisée par loop
symbol masque = w4              ' variable bit de masquage
symbol var_in = w5              ' variable données utilisée pendant shiftin
symbol var_out = w6       'variable données utilisée pendant shiftout
symbol bits = 8           ' nombre de bits
symbol MSBvalue = 128     ' MSBvalue
                                '(=128 pour 8 bits, 512 pour 10 bits, 2048 pour 12 bits)


'=====================================================================
' ~~~~~ SHIFTIN ROUTINES ~~~~~
' Une seule de ces 4 est nécessaire – voir les spécifications de votre matériel
' '=====================================================================
' ***** Shiftin LSB first, Data Pre-Clock *****
shiftin_LSB_Pre:
let var_in = 0
for counter = 1 to bits           ' nombre de bits
var_in = var_in / 2               ' décalage à droite LSB first
if serdata = 0 then skipLSBPre
var_in = var_in + MSBValue        ' set MSB si serdata = 1
skipLSBPre: pulsout sclk,1 ' Impulsion clock pour obtenir le prochain bit de donnée
next counter
return
'=====================================================================
' ***** Shiftin MSB first, Data Pre-Clock *****
shiftin_MSB_Pre:
let var_in = 0
for counter = 1 to bits           ' nombre de bits
var_in = var_in * 2               ' décalage à gauche MSB first
if serdata = 0 then skipMSBPre
var_in = var_in + 1               ' set LSB si serdata = 1
skipMSBPre: pulsout sclk,1 ' Impulsion clock pour obtenir le prochain bit de donnée
next counter
return
'=====================================================================
' ***** Shiftin LSB first, Data Post-Clock ***** '
shiftin_LSB_Post: let var_in = 0
for counter = 1 to bits       ' nombre de bits
var_in = var_in / 2           ' décalage à droite LSB first
pulsout sclk,1 '   Impulsion clock pour obtenir le prochain bit de donnée
if serdata = 0 then skipLSBPost
var_in = var_in + MSBValue ' set MSB if serdata = 1
skipLSBPost: next counter
return
'=====================================================================
' ***** Shiftin MSB first, Data Post-Clock *****
shiftin_MSB_Post: let var_in = 0
for counter = 1 to bits       ' nombre de bits
var_in = var_in * 2           ' décalage à gauche MSB first
pulsout sclk,1    ' Impulsion clock pour obtenir le prochain bit de donnée
if serdata = 0 then skipMSBPost
var_in = var_in + 1           ' set LSB si serdata = 1
skipMSBPost: next counter
return
'=====================================================================
```

2.144 shiftout (spiout)

228

(Tous les PICAXEs)

Syntaxe:

SPIOUT sclk,sdata,mode,(data{/ bits}, {data{/ bits},...})

- sclk est une variable/constante qui spécifie la broche entrée/sortie utilisée en horloge (clock).
- sdata est une variable/constante qui spécifie la broche entrée/sortie utilisée en données (data).
- Mode est une variable/constante (0-7) qui spécifie le mode:

0	LSBFirst_L	(LSB first, idles low)
1	MSBFirst_L	(MSB first, idles low)
4	LSBFirst_H	(LSB first, idles high)
5	MSBFirst_H	(MSB first, idles high)

- Data est une variable/constante qui contient la donnée à envoyer.
- Bits est le le nombre optionnel de bits à transmettre. S'il est omis, le nombre par défaut est 8.

Information:

La commande **spiout** (**shiftout** est également accepté par le compilateur) est une méthode 'bit-bang' de communication SPI sur les X1 et X2 uniquement. Tous les autres PICAXEs doivent utiliser le programme exemple inclus pour bénéficier de la même fonctionnalité.

Pour une solution hardware avec les X1/X2 parts voir la commande **hspiout**.

Par défaut, 8 bits sont décalés dans la variable. Un nombre de bits différent (1 to 8) peut être défini par l'option / bits.

Par exemple, si l'on souhaite décaler sur 12 bits, le faire sur deux bytes, un byte décalant 8 bits et le second byte décalant 4 bits. Noter que si on utilise la méthode MSB first les bits sont décalés vers la gauche, et ainsi, si le décalage est de 4 bits ils doivent être les bits7-4 (et non 3-0). Avec la méthode LSB les bits sont décalés vers la droite.

Lors d'une connexion d'un périphérique SPI (par ex. EEPROM) se rappeler que le data-in de l'EEPROM se connecte à data-out du PICAXE, et vice et versa.

Certains PICAXEs n'ont pas de commande shiftout.

Toutefois la même fonctionnalité peut être trouvée en utilisant la procédure décrite ci-dessous.

Effets de l'augmentation de la fréquence d'horloge.

Augmenter le fréquence d'horloge augmente la fréquence d'horloge du SPI .

Exemple:

```
spiout 1,2,LSB_First, (b1 / 8) ' clock 8 bits from b1
```

shiftin/shiftout sur les PICAXEs avec des commandes natives:

Certains PICAXE n'ont pas de commande **shiftout**. Toutefois la même fonctionnalité peut être trouvée en utilisant la procédure décrite ci-dessous.

Ces procédures sont aussi placées dans le fichier nommé **shiftin_out.bas** dans le dossier **\samples** du logiciel Programming Editor.

Pour les utiliser, copier simplement les définitions de symboles en tête de votre programme et copier la procédure shiftout en bas du programme.

Ne pas copier toutes les options pour économiser de la mémoire.

Il est sous entendu que les sorties data et clock (sdata and sclk) sont à l'état bas avant d'utiliser le gosub.

La ligne BASIC

"shiftout sclk, sdata,mode, (var_out(\bits))"

devient

gosub shiftout_LSBFirst (for mode LSBFirst)

gosub shiftout_MSBFirst (for mode MSBFirst)

La definition des symboles de la commande **shiftin** doit être également utilisée.

```
'=====================================================================
' ***** Shiftout LSB first *****
shiftout_LSBFirst:
for counter = 1 to bits       ' nombre de bits
masque = var_out & 1          ' masque LSB
low sdata                     ' data low
if masque = 0 then skipLSB
high sdata ' data high
skipLSB: pulsout sclk,1 ' Impulsion clock for 10us
var_out = var_out / 2 ' décalage à droite de la variable pour LSB
next counter
return
'=====================================================================
' ***** Shiftout MSB first *****
shiftout_MSBFirst:
for counter = 1 to bits ' nombre de bits
masque = var_out & MSBValue ' masque MSB
high sdata ' data high
if masque = 0 then skipMSB
low sdata ' data low
skipMSB: pulsout sclk,1 ' Impulsion clock for 10us
var_out = var_out * 2 ' décalage à gauche de la variable pour MSB
next counter
return
'=====================================================================
```

2.145 Sleep

230

(Tous les PICAXEs)

Syntaxe:

SLEEP period

- Period est une variable/constante qui spécifie la durée de sommeil en un multiple de 2.3 seconds (1-65535).

Fonction:

Mise en sommeil pour une période multiples de 2.3s approximativement (2.1s sur les X1/X2).

Information:

La commande **sleep** met le microcontrôleur en mode de faible consommation pendant une certaine durée. Quand il est dans ce mode tous les timers sont stoppés et les commandes **servo** et **pwm** ne fonctionnent plus.

La période nominale est 2.3s, aussi **sleep 10** sera d'environ 23 seconds. La commande **sleep** n'est pas régulée et en raison des tolérances des timers internes du microcontrôleur cette durée est sujette à une tolérance de -50 to +100%. La température externe affecte également cette tolérance, donc aucun projet nécessitant de la précision ne devrait utiliser cette commande.

Des mises en sommeil plus courtes sont possibles avec la commande **nap** (si supporté).

Certains PICAXE supportent la commande **disablebod** (**enablebod**) pour désactiver la détection de chute de tension d'alimentation. Utiliser cette commande préalablement à **sleep** réduit considérablement le courant consommé pendant la commande **sleep**.

Sur les autres PICAXEs que les X2 la commande **sleep 0** est ignorée.

Sur les X2 **sleep 0** mets le microcontrôleur en sommeil permanent car il ne se réveille pas toutes les 2.1 secondes. Le microcontrôleur n'est réactivé que par interruption hardware ou par hard-reset. Le PICAXE ne répondra pas à une demande de téléchargement s'il est en sommeil permanent.

Effets de l'augmentation de la fréquence d'horloge.

La commande **sleep** utilise le chien de garde interne qui n'est pas lié à la fréquence du résonateur.

Exemple:

```
main:
      high 1          ' mets la sortie 1 à l'état haut
      sleep 10        ' sommeil pendant 23 secondes
      low 1           ' mets la sortie 1 à l'état bas
      sleep 100       ' sommeil pendant 230 secondes
      goto main       ' retour au début
```

2.146 sound

231

Syntaxe :

SOUND pin,(note,duration,note,duration...)

"pin" est une variable ou une constante qui spécifie la broche (port et broche si besoin)

"note" est une variable ou une constante (0-255) spécifie le genre et la fréquence de la note.

Note = 0 force le silence pour la durée spécifiée. Les notes 1-127 produisent des sons de ton croissant (de plus en plus aiguës), les notes 128-255 produisent des sons de type " bruit blanc " de ton croissant.

"duration" est une variable ou une constante qui fixe la durée de la note qui précède, approximativement un multiple de 10 ms.

Concerne :

08 / 08M / 08M2

14M / 14M2

18 / 18A / 18M / 18M2 / 18X

20M / 20M2 / 20X2

28A / 28X / 28X1 / 28X2

40X / 40X1 / 40X2

Commentaire :

Permet de générer des signaux sonores sous forme de bips à utiliser pour des jeux, des saisies claviers, etc.

Pour la musique, utiliser les instructions "play" ou "tune".

Remarques :

1/ Le couple "note,duration" est impératif.

2/ La durée de la note est divisée par 2 si l'on double la fréquence horloge (8 MHz) et divisée par 4 si on la quadruple (16 MHz).

Exemple:

```
b0=20
sound B.7,(b0,50)
```

2.147 srlatch

232

Syntaxe:

SRLATCH config1, config2

- Config1 est une variable/constante qui spécifie la configuration de la bascule (latch)

Bit 7 = 1 SR Latch est active
= 0 SR Latch n'est pas utilisée
Bit 6-4 SR Clock bits de division – configure la fréquence d'horloge de la bascule SR

654	Divider	16MHz	8MHz	4MHz
000	1/4	0.25us	0.5us	1us
001	1/8	0.5	1	2
010	1/16	1	2	4
011	1/32	2	4	8
100	1/64	4	8	16
101	1/128	8	16	32
110	1/256	16	32	64
111	1/512	32	64	128

Bit 3 = 1 Q est présent sur la broche SRQ (quand elle est une sortie)
= 0 Broche SRQ n'est pas utilisée sur le module SR Latch
Bit 2 = 1 NON Q est présent sur la broche broche SRNQ (quand elle est une sortie)
= 0 Broche SRNQ n'est pas utilisée sur le module SR Latch
Bit 1 = 0 Non utilisé, laisser à 0
Bit 0 = 0 Non utilisé, laisser à 0

Noter que tous les PICAXEs n'ont pas en même temps de broches SRQ et SRNQ. Certains n'ont que SRQ et d'autres que SRNQ. Voir les schémas de brochage.
Noter également que SRNQ sur les 28X2/40X2 est la broche sertxd programming. Les commandes debug et sertxd ne peuvent fonctionner si SRNQ est active (via bit 2).

- Config2 est une variable/constante qui configure le set/reset

Quand le bit est 0, il n'y a pas d'effet sur le SR latch.

Pour les 20X2 :

Bit 7 = 1 HINT1 met à 1 la bascule (voir hintsetup)
Bit 6 = 1 broche Latch set est liée à l'horloge (voir au dessus)
Bit 5 = 1 C2 comparator met à 1 la bascule (voir compsetup)
Bit 4 = 1 C1 comparator met à 1 la bascule (voir compsetup)
Bit 3 = 1 HINT1 met à 0 la bascule (voir hintsetup)
Bit 2 = 1 broche Latch reset est liée à l'horloge (voir au dessus)
Bit 1 = 1 C2 comparator met à 0 la bascule (voir compsetup)
Bit 0 = 1 C1 comparator met à 0 la bascule (voir compsetup)

Pour les 28X2/40X2:

Bit 7 = 1 broche SRI high met à 1 la bascule
Bit 6 = 1 broche Latch set est liée à l'horloge (voir au dessus)
Bit 5 = 1 C2 comparator met à 1 la bascule (voir compsetup)
Bit 4 = 1 C1 comparator met à 1 la bascule (voir compsetup)
Bit 3 = 1 broche SRI high met à 0 la bascule
Bit 2 = 1 broche Latch reset est liée à l'horloge (voir au dessus)
Bit 1 = 1 C2 comparator met à 0 la bascule (voir compsetup)
Bit 0 = 1 C1 comparator met à 0 la bascule (voir compsetup)

Noter que sur les 28X2/40X2 la broche SRI peut agir soit en set ou en reset selon les bit 3 ou bit 7. Ne pas mettre ces bits à 1 simultanément.

Pour les M2:

Bit 7 = 1 broche SRI à 1 met à 1 la bascule
Bit 6 = 1 broche Latch set est liée à l'horloge (voir au dessus)
Bit 5 = 0 Non utilisé, laisser à 0
Bit 4 = 0 Non utilisé, laisser à 0
Bit 3 = 1 broche SRI à 1 met à 0 la bascule
Bit 2 = 1 broche Latch reset est liée à l'horloge (voir au dessus)

(c) Revolution Education Ltd.
All rights reserved

Bit 1 = 0 Non utilisé, laisser à 0
Bit 0 = 0 Non utilisé, laisser à 0

Noter que sur les M2 la broche SRI la broche SRI peut agir soit en set ou en reset selon les bit 3 ou bit 7. Ne pas mettre ces bits à 1 simultanément.

Fonction:

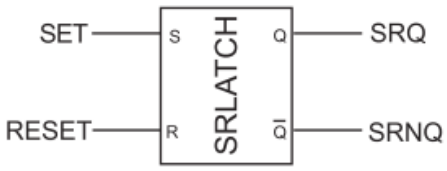
Configure la bascule SR hardware interne. La bascule peut être mise à 1 par la commande SRSET, ou l'un des périphériques listés ci dessus. De la même manière la bascule peut être mise à 0 par la commande SRRESET commande ou l'un des périphériques. Si les signaux SET et RESET sont présents, la bascule est mise à l'état reset (0)

Information:

Certains PICAXEs ont une bascule interne SR physique. Cette bascule peut être utilisée indépendamment du programme PICAXE, ainsi, par exemple, une sortie peut être instantanément controlée par la bascule.
La bascule SR dispose également d'une horloge interne. Cela signifie que la bascule peut être configurée pour agir comme un timer 555.
La sortie (Q) peut être disponible sur la broche SRQ (si présente). La sortie inverse (NOT Q) peut être disponible sur la broche SRNQ (si présente).
La commande srlatch ne configure pas automatiquement ces broches comme sorties. Cela doit être fait auparavant dans le programme.

Exemple pour 20X2:

```
init: low B.1
high C.4
srlatch %10001100, %00000000
main: srset      ; bascule à 1
pause 5000
srreset          ; bascule à 0
pause 5000
goto main        ; retour au début
```



2.148 srset / srreset

234

Syntaxe:

SRSET

SRRESET

Fonction:

Active ou désactive la bascule SR physique.

Information:

Ces deux commandes peuvent activer ou désactiver la bascule SR via le programme du PICAXE. Noter que la bascule SR peut aussi être configurée par des périphériques physiques.

- Voir la commande SRLATCH pour plus de détails.

Exemple pour 20X2:

```
init: low B.1
high C.4
srlatch %10001100, %00000000

main:
        srset
        pause 5000
        srreset
        pause 5000
        goto main          ; retour au début
```


(c) Revolution Education Ltd.
All rights reserved

2.149 stop

235

Syntaxe:

STOP

Concerne :

08 / 08M / 08M2

14M / 14M2

18 / 18A / 18M / 18M2 / 18X

20M / 20M2 / 20X2

28A / 28X / 28X1 / 28X2

40X / 40X1 / 40X2

Commentaire :

Crée une boucle permanente à la fin du programme.

On ne peut l'interrompre qu' en faisant un reset par coupure-rétablissement de l'alimentation ou en rechargeant le programme à partir de l'ordinateur.

Remarques :

Le circuit ne passe pas en mode veille (low power mode).

Les temporisateurs internes (timers) ne sont pas arrêtés si bien que les fonctions "servo" et "pwmout" continuent à fonctionner.

Exemple:

```
main:
      pwmout C.1,120,400
stop                              ; boucle permanente
```


(c) Revolution Education Ltd.
All rights reserved

2.150 suspend

236

Syntaxe :

SUSPEND tâche

- Tâche est une variable, ou constante, spécifiant quelle tâche doit être suspendue.

Fonction:

Interrompt momentanément une tâche.

Information :

Les puces type M2 peuvent faire tourner plusieurs tâches en paralèlle. La commande 'suspend' permet d'interrompre momentanément une tâche. Toutes les autres tâches continuent de fonctionner normalement. Si la tâche est déjà en cours de fonctionnement, la commande est ignorée (NdT : Erreur, lire "Si la tâche est déjà suspendue, la commande est ignorée") . Si vous souhaitez une mise en pause de votre programme, lors d'un reset, utilisez la commande 'suspend' dans la première ligne . La puce s'arrêterra d'elle même lors du reset.

Ne pas suspendre toutes les tâches en même temps !

Exemple:

```
start0:
        high B.0          ; B.0 à 1
        pause 100         ; attente 0.1 seconde
        low B.0           ; B.0 à 0
        pause 100         ; attente 0.1 seconde
        goto start0       ; boucle

start1:
        pause 5000        ; attente 5 secondes
        suspend 0         ; suspendre la tâche 0
        pause 5000        ; attente 5 secondes
        resume 0          ; retour à la tâche 0
        goto start1       ; boucle
```

2.151 swap

237

Syntaxe:

SWAP variable1, variable2

Commentaire :

Echange les valeurs contenues dans deux variables.

Exemple:

```
b1 = 5
b2 = 10
swap b1,b2    ; maintenant b1 = 10 et b2 = 5
```


(c) Revolution Education Ltd.
All rights reserved

2.152 switch on/off

238

Syntaxe :

SWITCH ON broche, broche, broche ...

SWITCHON broche, broche, broche ...

SWITCH OFF broche, broche, broche ...

SWITCHOFF broche, broche, broche ...

- Broche est une variable, ou constante, spécifiant quelle(s) broches(s) est(sont) concernée(s)

Fonction:

Bascule broche 1/0

Information :

Il s'agit d'une 'pseudo commande' destinée au plus jeunes étudiants, et équivalente aux 'high' et 'low'

Exemple:

```
main: switch on 7          ; bascule la sortie 7 à 1
      wait 5               ; attente 5 secondes
      switch off 7         ; bascule la sortie 7 à 0
      wait 5               ; attente 5 secondes
      goto main            ; boucle
```


(c) Revolution Education Ltd.
All rights reserved

2.153 symbol

239

(tous les PICAXEs)

Syntaxe:

SYMBOL nom_symbole = valeur

SYMBOL nom_symbole = valeur ?? constante

- Nom_symbole est une chaîne de caractères qui doit commencer par un caractère alphabétique ou '_'.
- A la suite du premier caractère il est possible d'utiliser des chiffres ('0'-'9').
- Valeur est une variable ou une constante affectée à nom_symbole.
- ?? peut être une fonction mathématique supportée, par exemple + - * / etc.

Fonction:

Assigne une valeur à nom_symbole

Des opérateurs mathématiques peuvent être utilisés avec des constantes. (non autorisé avec des variables)

Information:

Les symboles sont utilisés pour nommer des constantes ou des variables afin de les rendre plus facilement identifiables dans un programme.

Les symboles n'ont aucune conséquence sur la longueur du programme, car ils sont convertis en nombres avant le chargement.

Bien entendu, le nom_symbole ne peut être un nom de commande, ou un mot réservé tel que input, step, etc.

(Voir la liste des mots réservés en annexe.)

Pour les entrées /sorties bien faire attention d'utiliser le terme 'pin0' et non '0' pour décrire une variable utilisée avec if...then .

Exemple:

```
symbol RED_LED = B.7          ; définit une sortie
symbol PUSH_SW = pinC.1       ; définit un inter d'entrée
symbol DELAY = b0             ; définit un symbole pour une variable
let DELAY = 200               ; pré charge un compteur par 200
main:
      high RED_LED            ; met la sortie 7 à 1
      pause DELAY             ; attend 0.2 seconde
      low RED_LED             ; met la sortie 7 à 0
      pause DELAY             ; attend 0.2 seconde
goto main                     ; retour au début
```

Complément (NdT):

Une broche de PICAXE peut selon les cas se nommer C.3 ou pinC.3.
On peut par exemple associer un **Symbol Led1 = C.3** et l'utiliser en sortie par un **HIGH Led1**
(Le fait d'utiliser l'instruction **HIGH** ayant deux effets: configurer automatiquement la broche C.3 en sortie, et la mettre à un niveau haut.)

Indépendamment de cela, si auparavant on a Définit C.3 en tant que sortie par la commande **DirsC = %00001000**, il est alors possible de de commander cette sortie par une instruction **LED1 = 1** à condition d'avoir associé le symbole par un **Symbol LED1 = pinC.3**

Concernant les entrées:
En supposant que l'on affecte : **Symbol Poussoir = C.2**

(c) Revolution Education Ltd.
All rights reserved

Lorsqu'on fait un test du type **IF Poussoir = 1 then ...** ça indiquera une erreur.

Par contre avec **Symbol Poussoir = pinC.2**, **IF Poussoir = 1 then ...** sera accepté.

En fait il faut considérer que C.2 est la broche et pinC.2 en est le contenu (1 ou 0).

Ceci permet d'écrire par exemple : **Led1 = Poussoir**
au lieu de :

```
IF Poussoir  THEN
HIGH Led1
Else
Low Led1
Endif
```

L'instruction **HIGH** peut être suivie d'une variable ...
par exemple **HIGH b1** sera équivalent à **HIGH C.1** si b1 a été définie comme étant égal à 9.
Soit

```
Symbol  sortie = b2
For sortie = 0 to 15
High sortie          'va mettre à 1 les sorties B.0 à C.7
```

En fait, vu du Pic les broches B.0 à B.7 puis C.0 à C.7, puis A.0 à A.7, puis D.0 à D.7 sont vues comme les sorties 0 à 31
(Selon les ports disponibles sur le PICAXE…)

(c) Revolution Education Ltd.
All rights reserved

2.154 table

240

Syntaxe :

TABLE {emplacement},(donnée, donnée...)

- Emplacement est une constante, optionelle spécifiant où stocker la donnée dans le tableau de mémorisation du programme. Si cet élément n'est pas spécifié, le stockage se poursuivra jusqu'au premier emplacement contenant zéro.

  Données sont des constantes type octets (0-255) mémorisées dans le tableau.

Fonction:

Construit un tableau de données, consultables depuis le programme.

Les puces type M2 disposent de 512 emplacements (0-511). Les autres puces disposent de 256 emplacements (0-255).

Information :

Il ne s'agit pas exactement d'une commande mais d'une technique, de pré chargement d'un tableau de valeurs utilisables par le programme.

Ces données pourront être lues via la commande 'readtable' elles seront figées, et ne pourront être altérées par un téléchargement).

La commande 'tablecopy' peut être utilisée, pour copier ces données en RAM.

Exemple:

```
TABLE 0,("Hello World")                ; sauve les données dans la table
main:
        for b0 = 0 to 10               ; démarrage boucle
                readtable b0,b1        ; lecture valeur dans la table
                serout 7,N2400,(b1)    ; transmission vers LCD module (série)
        next b0                        ; suite
```


(c) Revolution Education Ltd.
All rights reserved

2.155 tablecopy

241

Syntaxe :

TABLECOPY début_emplacement, bloc_taille

- Début_emplacement est le numéro du premier élément à copier (0-511)
- Bloc_taille est le nombre d'octets à copier en RAM (1-512)

Fonction:

Copie le tableau de données en RAM. Chaque adresse est directement recopiée.(Les adresses correspondent).

Information :

La commande tablecopy peut être utilisée, pour copier rapidement un tableau de données dans la RAM.Comme chacune de ces copies est effectuée exactement à la même adresse, elles peuvent sans problème être utilisées par les instructions 'peek' ou '@bptr'.

La copie ne fonctionnera que jusqu'à l'adresse maximum de la table (soit 511).

Attention à ne pas effacer les variables 'basses' (b0, b1, b2, etc) en utilisant cette instruction. (mêmes emplacements en RAM).

Exemple:

```
TABLE 0,("Hello World")          ; sauve les données dans le tableau

main:
        tablecopy 0,5            ; copie les adresses 0,1,2,3,4
        debug                    ; transfert b0-b4 sur PC
        goto main                ; boucle
```


(c) Revolution Education Ltd.
All rights reserved

2.156 tmr3setup

242

Syntaxe :

TMR3SETUP configuration

- Configuration est une variable ou une constante qui va configurer le timer 3.

Configuration est définite comme suit (20X2, 28X2,-5V,28X2-3V,40X2-3V, 40X2-5V) :

Bit 7	doit être positionné à	(1)
Bit 6	doit être positionné à	(0)
Bit 5, 4	1 : 8 pré-positionnement	(11)
	1 : 4 pré-positionnement	(10)
	1 : 2 pré-positionnement	(01)
	1 : 1 pré-positionnement	(00)
Bit 3	Doit être à zéro	(0)
Bit 2	Doit être à zéro	(0)
Bit 1	Doit être à zéro	(0)
Bit 0	Validation timer 3	(1 = on, 0 = off)

Configuration est définite comme suit (28X2, 40X2) :

Bit 7	doit être positionné à	(0)
Bit 6	doit être positionné à	(0)
Bit 5, 4	1 : 8 pré-positionnement	(11)
	1 : 4 pré-positionnement	(10)
	1 : 2 pré-positionnement	(01)
	1 : 1 pré-positionnement	(00)
Bit 3	doit être positionné à	(0)
Bit 2	doit être positionné à	(0)
Bit 1	doit être positionné à	(1)
Bit 0	Validation timer 3	(1 = on, 0 = off)

Fonction:

Configuration des modes de fonctionnement du timer3 sur les puces X2)

Information :

La commande tmr3setup configure le timer3 sur les puces X2 . Il s'agit d'un compteur, qui fonctionne en arrière plan, et peut donc être utilisé en tâche de fond.

La vitesse de fonctionnement du timer est de (1/vitesse d'horloge) * 4

En conséquence, dans le cas d'une vitesse de 8 Mhz, le timer s'incrémentera toutes les 0.5µs. Il est possible de modifier cette durée (via le positionnement des bits 5 et 4) d'une plage de 1 :1 à 1 :8, dans ce dernier cas, le timer s'incrémentera toutes les 4µs (8 * 0.5µs).

La variable interne (word) au PICAXE s'incrémentera à chaque dépassement du timer (65536 passages). Donc, à 8 Mhz, avec un facteur huit, timer3 s'incrémentera toutes les 262144µs, soit 262ms. 'timer3' est de type word.

Exemple (pour 28X2):

```
tmr3setup %00110011          ; timer3 on, 1:8 prépositionnement
main: pause 500              ; petit délai
      debug                  ; affiche valeur timer3 (PC)
      goto main
```

Exemple (pour 28X2-5V ou 28X2-3V):

```
tmr3setup %10110001          ; timer3 on, 1:8 prépositionnement
main: pause 500              ; petit délai
      debug                  ; affiche valeur timer3 (PC)
      goto main
```

Exemple (le code permet le choix automatique entre 28X2, 28X2-3V, ou 28X2-5V):

```
readsilicon b1               ; récupération type de puce
b1 = b1 & %11100000          ; masquage des 4 bits de type
if b1 = %10000000 then       ; la puce est un 28X2
      tmr3setup %00110011    ; timer3 on, 1:8 prépositionnement
else                         ; autre type de puce
      tmr3setup %10110001    ; timer3 on, 1:8 prépositionnement
endif

main: pause 500              ; petit délai
      debug                  ; affiche valeur timer3 (PC)
      goto main
```


(c) Revolution Education Ltd.
All rights reserved

2.157 toggle

244

(tous les PICAXEs)

Syntaxe:

TOGGLE pin,pin,pin...

- Pin est une variable / contante qui definit la broche concernée.

Fonction:

Configure une broche en sortie et inverse son état logique.

Information:

La commande Toggle inverse l'état logique (high si low et vice versa)

Sur les microcontrôleur avec des entrées sorties configurables (PICAXE-08 par ex.) cette commande configure automatiquement les broches en sorties.

Exemple:

```
main:
toggle B.7      ; inverse la sortie 7
pause 1000      ; attend 1 seconde
goto main       ; retour au début
```


(c) Revolution Education Ltd.
All rights reserved

2.158 togglebit

245

Syntaxe:

TOGGLEBIT var, bit

"var" est la cible (octet ou mot)
"bit" est la broche à modifier

Concerne :

20X2

28X1 / 28X2

40X1 / 40X2

Commentaire :

Inverse l' état d'une broche . Si elle est en 1 la force en 0 et inversement

.

Exemples:

```
togglebit b6, 0     ;inverse l' état du bit 0 de b6
togglebit w4, 15    ;inverse l' état du bit 15 de w4
```


(c) Revolution Education Ltd.
All rights reserved

2.159 touch

246

Syntaxe:

TOUCH canal,variable

- Canal est une constante ou une variable spécifiant l'entrée ADC utilisée
- Variable est du type byte et reçoit la valeur lue par la commande touch

Fonction:

Lit un capteur relié à une broche compatible "ADC" et enregistre la valeur lue dans le byte "variable".

Cette commande configure la broche utilisée à la fois comme un capteur et un convertisseur ADC.

En réalité, touch est une pseudo commande modifiant la commande touch16 pour la rendre compatible avec une variable de type byte. Cette commande peut être suffisante dans des programmes simples, mais la résolution est réduite par cette conversion.

Il est donc recommandé d'utiliser la commande touch16, qui garde une résolution maximum et offre des possibilités plus étendues.

Avant utilisation, débrancher les câbles de programmations qui pourraient perturber la mesure.

Le câblage interne et externe de chaque broche est particulier, la calibration doit se faire individuellement.

En termes simples, la commande touch est un capacimètre, la valeur retournée dans "variable" dépend de cette capacité, mesurée entre le 0v et le capteur.

Voir la commande touch16 pour plus de détails.

Effets de la vitesse du microcontrôleur:

Le résultat de la mesure dépend de la vitesse oscillateur choisie. Chaque vitesse donnera un résultat différent. La calibration des capteurs dépend donc de ce choix.

Exemple:

```
Debut:
  Touch C.1,b0          ;le résultat de la lecture est dans b0
  If b0 > 100 then      ;test sur b0>100
     High B.2           ;si oui => sortie B.2 ON
  Else                  ;sinon
     Low B.2            ;sortie B.2 OFF
  Endif                 ;fin de test
  Goto debut            ;fin de boucle
```


(c) Revolution Education Ltd.
All rights reserved

2.160 touch16

247

Syntaxe:

TOUCH16, canal, wordvariable

TOUCH16, [configuration], canal, wordvariable

- Canal est une constante ou une variable spécifiant l'entrée ADC utilisée
- wordvariable est du type word et reçoit la valeur lue par la commande touch (16 bits pour série M2, 10bits pour X2)
- Configuration est une constante ou une variable optionnelle [entre crochets] spécifiant le byte de configuration.

Fonction:

Lit un capteur relié à une broche compatible "ADC" et enregistre la valeur lue dans une variable type word: "wordvariable".

Cette commande configure la broche utilisée à la fois comme un capteur et un convertisseur ADC.

Information:

La commande touch16 est utilisée pour lire un capteur (ex capteur de toucher) relié a une broche compatible ADC/touch du microcontrôleur.

IMPORTANT – Ne pas toucher directement la broche avec les doigts ou une pièce métallique. Pour une fonction touche, la broche doit être reliée, sans résistance de tirage, à une surface métallique électriquement isolée, d'une surface d'environ 15mm de diamètre, recouverte d'un isolant, par exemple une feuille de plastique de 2mm d'épaisseur, ou dessinée sur un circuit imprimé et les touches dessinées du côté isolant, ou constituées de plaques métalliques sous l'épaisseur d'un boitier, etc…Bien visualiser l'emplacement des touches en laissant un espacement suffisant.

Avant utilisation, débrancher les câbles de programmations (AXE026) qui pourraient perturber la mesure.

Le câblage interne et externe de chaque broche est particulier, la calibration doit se faire individuellement.

En termes simples, la commande touch est un capacimètre, la valeur retournée dans "variable" dépend de cette capacité, mesurée entre le 0v et le capteur. L'approche d'un doigt augmente la capacité parasite lue et la valeur retournée dans la variable augmente.

Un défaut d'isolement ou un dépassement de la capacité maximum donne une variable égale à 0. Une valeur nulle est toujours une anomalie.

Ces capteurs sont sensibles à l'humidité et doivent être gardé au sec.

Bien programmés, les capteurs utilisant cette commande remplacent poussoirs, interrupteurs, voire potentiomètres, avec un "design" élégant et un coût imbattable.

Voir la démonstration de la carte AXE181.

Les PICAXEs série M2 et X2 utilisent des méthodes de mesure différentes. Les X2 sont plus rapides mais ont une résolution plus faible (10 bits au lieu de 16).

Effets de la vitesse du microcontrôleur:

Le résultat de la mesure dépend de la vitesse oscillateur choisie. Chaque vitesse donnera un résultat différent. La calibration des capteurs dépend donc de ce choix.

Exemple:

```
Debut:
  Touch C.1,w0          ;le résultat de la lecture est dans b0
  If w0 > 3000 then     ;test sur w0
     High B.2           ;si oui => sortie B.2 ON
  Else                  ;sinon
     Low B.2            ;sortie B.2 OFF
  Endif                 ;fin de test
```

```
Goto debut              ;fin de boucle
```

Octet de configuration pour PICAXEs série M2

Normalement, il est recommandé de conserver la configuration par défaut. Dans ce cas, l'octet de configuration n'est pas nécessaire dans la commande.

Toutefois, la configuration par défaut peut être modifiée en insérant l'octet de configuration entre crochets (sans espaces). Pour la série M2, cet octet est divisé en trois parties:

Octet par défaut = %000 01 001

Bit7,6,5 = Valeur initiale du compteur

= 000 => Nombre d'oscillations requises = 256 (valeur par défaut)

= 010=>Nombres d'oscillations requises = 192

= 100=>Nombres d'oscillations requises = 128

= 110=>Nombres d'oscillations requises = 64

= 111=>Nombres d'oscillations requises = 32

Note du traducteur: En fait, cette option divise la valeur retournée dans la variable, si 000 donne 4000, 100 donne 2000

Bit 5,4 = Gamme de mesure

= 00 Arrêt de l'oscillateur commande touch16

= 01 Gamme basse (0,1µA) (Valeur par défaut)

= 10 Gamme moyenne (1,2µA)

= 11 Gamme haute (18µA)

Note du traducteur: Cette option diminue la sensibilité du capteur et diminue la valeur de la variable.

Bit 2,1,0 = Diviseur du compteur (divise par 2, jusqu'à 256

= 001=>Division par 4 (valeur par défaut)

Note du traducteur: En fait, la modification de la valeur par défaut multiplie la valeur de la variable.

Octet de configuration pour PICAXE série X2

Normalement, il est recommandé de conserver la configuration par défaut. Dans ce cas, l'octet de configuration n'est pas nécessaire dans la commande.

Toutefois, la configuration par défaut peut être modifiée en insérant l'octet de configuration entre crochets (sans espaces). Pour la série M2, cet octet est divisé en trois parties:

Octet par défaut = %00 11 0010

Bit 7,6 = Inutilisés

Bit 5,4 = 00 Arrêt de l'oscillateur commande touch16

= 01 Courant de charge nominal

= 10 Courant de charge moyen (10 x nominal)

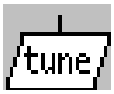
= 11 Courant de charge fort (100 x nominal) (valeur par défaut)

Bit 3,2,1,0 = Modifie le temps de charge par multiple de 2µs (de 1à 15) (valeur par défaut = 2)

(c) Revolution Education Ltd.
All rights reserved

2.161 tune

250



Syntaxe :

TUNE pin, speed, (note, note, note...)

TUNE pin, speed, LED_mask, (note, note, note...) *(modèles* M2 *seulement)*

TUNE LED_option, speed, (note, note, note...) *(modèles* 08 *seulement)*

- "pin" est une variable ou une constante qui spécifie la sortie à utiliser.4

  Sur les modèles 08 cette sortie ne peut être que la sortie 2 (broche 5).
- "speed" est une variable ou une constante (1-15) qui fixe le tempo (rythme).
- "notes" sont les notes de la mélodie générées par le "Tune Wizard" (Assistant de composition).
- "LED_mask" (modèles M2 seulement) est une variable ou une constante qui entraîne une ou plusieurs sorties (sur le même port que celui choisi pour la sortie "pin") à basculer (0/1) au rythme de la mélodie. Par exemple %00000011 basculera les sorties 0 et 1 de ce port.
- "LED_option" (sur les modèles 08 seulement) est une variable ou une constante (0-3) qui entraîne une ou plusieurs sorties à basculer (0/1) au rythme de la mélodie. Les options sont :

  0 : pas de sorties

  1 : bascule la sortie 0 (broche 7) on et off

  2 : bascule la sortie 4 (broche 3) on et off

  3 : bascule les sorties 0 et 4 alternativement

Concerne :

08M / 08M2

14M / 14M2

18M /18M2

20M / 20M2 / 20X2

28X1 / 28X2

40X1 / 40X2

Commentaire :

L' instruction "tune" permet de jouer une mélodie programmée. La musique jouée avec un micro-contrôleur, pour lequel la capacité mémoire est limitée, n' aura pas la qualité des appareils spécialisés du commerce mais l' instruction "tune" donne des résultats remarquables. La musique peut être écoutée sur des hauts-parleurs piézoélectriques peu onéreux (comme ceux qui équipent

les cartes de vœux) ou sur des hauts-parleurs électrodynamiques de meilleure qualité.

Les mélodies programmées sont jouées note aprés note, une seule à la fois (musique monophonique).

Les informations qui suivent expliquent comment encoder les notes. Vous pouvez utiliser l' assistant " Tune Wizard " pour générer automatiquement les attributs de l' instruction "tune" ou coder les notes successives manuellement ou encore en important les mélodies contenues dans un téléphone portable par exemple.

Les informations techniques ci-dessous ne sont donc données qu' à titre indicatif, elles ne sont pas nécessaires pour utiliser l'assistant " Tune Wizard ".

Remarques :

1/ L' instruction "tune" compresse les données mais plus la mélodie dure longtemps plus elle utilise d'emplacements mémoire. Par contre , l' instruction "play" n' occupe pas d'emplacements mémoire supplémentaires mais est limitée à quatre mélodies pré-enregistrées.

2/ Toutes les mélodies peuvent être écoutées sur un haut-parleur piézoélectrique ou électrodynamique connecté sur la sortie choisie (uniquement la sortie 2 (broche 5) sur les modèles 08). Des exemples de câblage utilisables sont donnés ci-dessous.

3/ Sur les modèles 08 et tous les M2 d'autres sorties peuvent être utilisées pour basculer au rythme de la mélodie. Les attributs "LED_option" (modèles 08) et "LED_mask" (modèles M2) basculent ces sorties "on/off" à la fin de chaque note.

"Speed" :

"speed" est une variable ou une constante qui fixe le tempo (rythme) de la mélodie. Le tempo est le nombre de battements par minute (BPM).

"speed" peut prendre les valeurs (1-15) et les valeurs des BPM correspondantes sont données par le tableau ci-joint (valeurs arrondies) qui correspondent à la plupart des mélodies usuelles.

La durée totale d'un battement est la somme de la durée de la note (8/9 ème du temps total) suivie de la durée d'un silence (1/9 ème du temps total).

La durée totale de la note est : speed * 65,64 ms (ms = milliseconde)

La durée du silence est : speed * 8,20 ms

La durée totale d'un battement est donc : (speed*65,64)+(speed*8,20) = (speed * 73,84) ms.

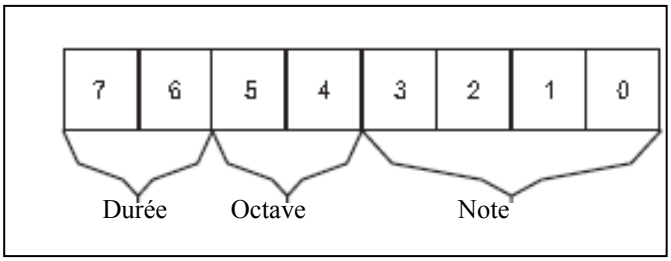
Le nombre de battements par minute (BPM) est donc 60000 / (speed * 73,84).

Remarque : Dans les instruments de musique électronique du commerce, la durée de la note dure les 7/8 ème de la durée du battement et le silence 1/8 ème. Avec un PICAXE ces rapports sont légèrement différents : 8/9 et 1/9.

Vitesse	BPM
1	812
2	406
3	270
4	203
5	162
6	135
7	116
8	101
9	90
10	81
11	73
12	67
13	62
14	58
15	54

" note "

Chaque note (" note ") est codée sur 8 bits (un octet).



- les bits 7 et 6 codent la durée :
  - 00 ---> ¼
  - 01 ---> 1/8
  - 10 ---> 1
  - 11 ---> ½

- les bits 5 et 4 codent l' octave
  - 00 ---> octave 6
  - 01 ---> octave 7
  - 10 ---> octave 5
  - 11 ---> non utilisé

- Les bits 3,2,1,0 codent la note (notation anglo-saxonne)
  - 0000 ---> C (---> Do)
  - 0001 ---> C#
  - 0010 ---> D (---> Ré)
  - 0011 ---> D#
  - 0100 ---> E (---> Mi)
  - 0101 ---> F (---> Fa)
  - 0110 ---> F#
  - 0111 ---> G (---> Sol)
  - 1000 ---> G#
  - 1001 ---> A (---> La)
  - 1010 ---> A#
  - 1011 ---> B (---> Si)

(c) Revolution Education Ltd.
All rights reserved

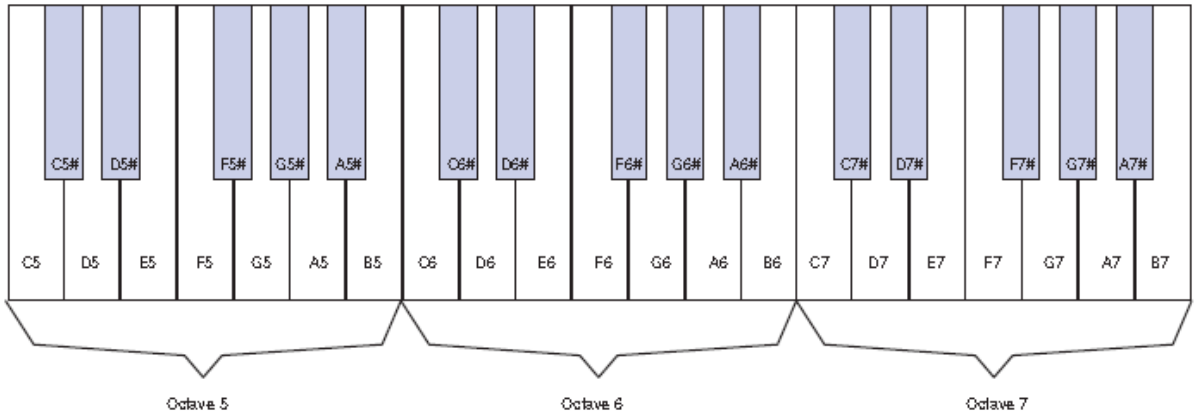
11xx ---> Pause

Par exemple %00001001 jouera la note A (La) (bit3 = 1,bit2 = 0, bit1= 0,bit0 = 1)

dans l' octave 6 (bit5 = 0, bit4 = 0)

pendant ¼ de la durée du tempo (bit7 = 0, bit6 = 0)

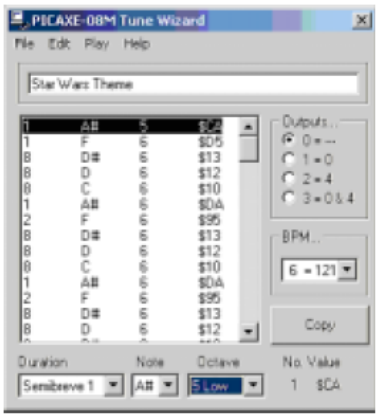
Emplacement des notes sur le clavier d'un piano :



Fréquence des notes

C5 = 262 Hz	C6 = 523 Hz	C7 = 1047 Hz
C5# = 277 Hz	C6# = 554 Hz	C7# = 1109 Hz
D5 = 294 Hz	D6 = 587 Hz	D7 = 1175 Hz
D5# = 311 Hz	D6# = 622 Hz	D7# = 1245 Hz
E5 = 330 Hz	E6 = 659 Hz	E7 = 1318 Hz
F5 = 349 Hz	F6 = 698 Hz	F7 = 1396 Hz
F5# = 370 Hz	F6# = 740 Hz	F7# = 1480 Hz
G5 = 392 Hz	G6 = 784 Hz	G7 = 1568 Hz
G5# = 415 Hz	G6# = 831 Hz	G7# = 1661 Hz
A5 = 440 Hz	A6 = 880 Hz	A7 = 1760 Hz
A5# = 466 Hz	A6# = 932 Hz	A7# = 1865 Hz
B5 = 494 Hz	B6 = 988 Hz	B7 = 1975 Hz

L' assistant " Tune Wizard "



Cet assistant permet soit de créer des mélodies manuellement en utilisant des menus déroulants soit d'en importer par Internet.(sonneries de téléphone portable). Ces sonneries sont disponibles sur Internet au format RTTTL (utilisé sur la plupart des téléphones Nokia). Veuillez noter que les PICAXEs ne peuvent jouer qu'une seule note à la fois (monophonie), et donc de peuvent utiliser des mélodies polyphoniques.

Il existe plus de 1000 mélodies à charger gratuitement sur le site www.PICAXE.co.uk (onglet ' software ').

Démarrage de l' assistant.

Ouvrir Programming Editor (PE).

Pour démarrer l'assistant, utilisez le menu PICAXE>Wizard> Ring Tones Tunes.

Une nouvelle fenêtre s' ouvre.

1/ Importation.

Pour importer une mélodie depuis Internet, le plus simple est de la trouver sur une page Web contenant une mélodie " Ring Tone Tune ". Il en existe plusieurs milliers à télécharger gratuitement. Choisir la version RTTTL (*Ringing Tones Text Transfer Language* utilisé sur la plus-part des téléphones Nokia). .

Voir par exemple : www.PICAXE.com/RTTTL-Ringtones-for-tune-command/

Ouvrir une mélodie : Edition > selectionner tout > Edition > copier

Retourner dans l' assistant : Edition > coller. Ringstone

Jouer

(Pour importer une mélodie sauvegardée sous la forme d'un fichier texte : Fichier > Import Ringtone).

Remarque : PICAXE ne peut jouer qu' une seule note à la fois, il faut donc choisir des mélodies ' monophoniques'

2/ Programmation manuelle .

Donner un titre, choisir un tempo (Menu déroulant (Md) 'Tempo-BPM'), choisir si d'autres sorties doivent basculer au rythme des notes (cases à cocher LED).

Décrire la première note : Durée (Md 'Duration'), Note (Md 'note'), Octave (Md 'octave')

Ces valeurs sont reportées automatiquement dans la partie affichage de la fenêtre.

Edition > Insérer ligne

Décrire la seconde note … et ainsi de suite.

Pour supprimer une ligne : selectionner / Edition > Delete line

Terminer par 'Copier'.

Le code peut alors être copié dans le programme principal.

Exemple :

Titre : ***Au clair de la Lune***

Fixer tempo = 3 / Octave = 6

Entrer successivement, ligne après ligne, les codes suivants (pour duration = 2 choisir " Minim ½ ", pour duration =1 choisir " Semibreve 1/1 ")

(duration note octave)
2 C 6
2 C 6
2 C 6
2 D 6
1 E 6
1 D 6
2 C 6
2 E 6
2 D 6
2 D 6
2 C 6

L' onglet 'jouer' permet de tester sur l' ordinateur s' il est équipé d'une carte son et de hauts-parleurs (la mélodie ainsi jouée peut donner une mauvaise idée de ce qu' elle sera sur un PICAXE).

Remarque :

Le code hexadécimal de la note (voir le paragraphe ' note ' ci-dessus) est donné à droite de chaque ligne.

Par exemple : 2 C 6 $C0.

En effet en reconstituant l' octet 'durée-octave-note' on a (11-00-0000) soit %11000000 ce qui s' écrit $C0 en hexadécimal.

La méthode la plus simple pour importer une mélodie depuis Internet consiste à la copier depuis une page web. Sélectionnez la version RTTTL de la mélodie puis cliquez Edition/copier. Revenez à l'assistant "Tune Wizard" puis cliquez "Edition/Coller".

Pour récupérer une mélodie depuis un fichier texte, utilisez Fichier/Importer RingTone

La mélodie peut être testée en cliquant dans le menu "jouer". L'interpréteur de mélodie donnera une bonne idée du résultat final que vous pourrez obtenir sur un PICAXE, sans que ce soit exactement la même chose. Ceci est du aux différences importantes de méthodes utilisées pour générer les sons sur un ordinateur et sur un PICAXE.

Sur les ordinateurs les plus anciens, la génération de la mélodie peut prendre quelques secondes car cette génération sollicitte fortement la mémoire vive de l'ordinateur.

Dès que la mélodie est terminée, vous pouvez cliquer sur le bouton "Copier" pour la transférer vers le Presse-Papier, puis la coller dans votre programme.

Menus de l'assistant "Ring Tones Tunes" = "Tune Wizard "

Fichier	Nouveau	Créer une nouvelle mélodie
	Ouvrir	Ouvrir une mélodie préalablement enregistrée
	Enregistrer sous	Enregistrer une mélodie
	Importer Ringtone	Ouvrir une mélodie préalablement enregistrée sous forme de fichier texte.
	Exporter RingTone	Enregistrer une mélodie sous forme de fichier texte.
	Exporter wave	Enregister au format Windows wave (*.wav)
	Fermer	Fermer l'assistant
Edition	Insérer Ligne	Insère une nouvelle ligne dans la mélodie
	Supprimer Ligne	Supprimer une ligne
	Copier BASIC...	Copie les commandes BASIC vers le presse papier
	Copier RingTone	Copie la mélodie vers le presse papier
	Coller BASIC	Colle les commandes BASIC depuis le presse papier
	Coller RingTone	Colle la mélodie depuis le presse papier
Jouer		Joue la mélodie sur le haut parleur d el'ordinateur
Aide	Aide	Ouvre le fichier d'aide.

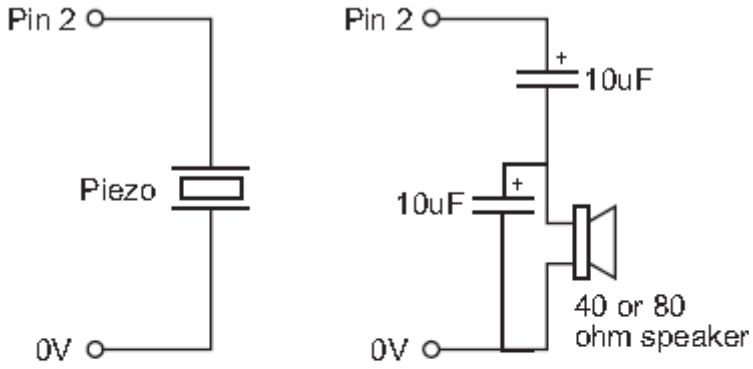
Trucs et astuces pour les mélodies

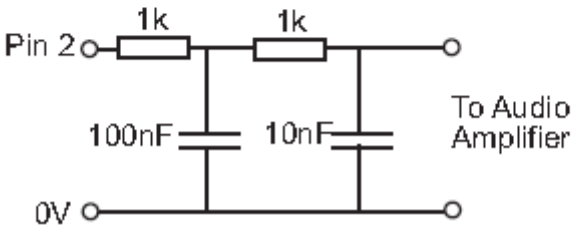
1. Après avoir généré une mélodie, essayez d'ajuster le tempo en augmentant ou en diminuant cette valeur puis en écoutant le résultat pour choisir ce qui donne le meilleur résultat.
2. Si votre mélodie ne s'importe pas, vérifiez que le nom de la mélodie (au début de la ligne) fait moins de 50 caractères et que tout le texte est enregistré sur une seule ligne.
3. Les mélodies qui contiennent l'instruction 'd=16' après leur description, ou qui contiennent beaucoup de notes commençant par 16 ou 32 ne se joueront pas correctement à la vitesse normale sur un PICAXE. Néanmoins, elles pourront être de meilleure qualité si vous utilisez la commande 'setfreqm8' juste avant de lancer la commande tune.
4. Le filtre d'importation de fichiers du PICAXE 'arrondi' les notes pointées (terminées par un '.') Vous pouvez souhaiter transformer ces notes en notes plus longues après l'importation.

Circuits électriques permettant la génération de sons :

La technique la plus simple et la plus économique pour jouer une mélodie est d'utiliser un transducteur piézo-électrique. Ils peuvent être connectés entre la broche de sortie choisie (par exemple pin 2 = cinquième broche d'un PICAXE 08M2) et le 0V. (Voir schémas ci-dessous) Les meilleures résultats seront obtenus avec les transducteur montés en boitier plastique. Les transducteurs nus sont souvent utilisés dans les écoles en raison de leur faible coût, mais leur coté cuivre nécessite d'être fixé sur une surface sonore (morceau de plastique, tasse en polystyrène choc, ou même sur le circuit imprimé) au moyen d'un adhésif double-face, ceci afin d'amplifier le son.	
Pour obtenir un meilleur son, l'utilisation d'un haut-parleur est indispensable. Encore une fois, la qualité du coffret dans lequel le haut-parleur est fixé est déterminante pour la qualité du son obtenu. Les haut-parleurs peuvent être activés "directement" au travers d'un condensateur en série, ou bien au moyen d'un amplificateur sommaire basé sur des transistors montés en Push-Pull Les haut-parleurs de 40 ou 80 ohms peuvent être branchés avec 2 condensateurs comme indiqué. Pour un haut-parleur de 8 ohms, utilisez la combinaison du haut-parleur et d'une résistance série de 33 ohms pour obtenir une résistance globale de l'ordre de 41 ohms. La sortie peut également être connectée (via un simple filtre RC) à un amplificateur audio comme le TBA820M.	

Les fichiers d'exemple sonore en .wav situés dans le sous-répertoire \music du Programming Editor sont des enregistrements réels de mélodies jouées (via un haut-parleur) à partir d'un PICAXE.





Format des fichiers : Ringing Tones Text Transfer Language (RTTTL):

<name> <sep> [<defaults>] <sep> <note-command>+
<name> := <char>+ ; max length 10 characters *PICAXE accepts up to 50*
<sep> := ":"
<defaults> :=
<def-note-duration> |<def-note-scale> |<def-beats>
<def-note-duration> := "d=" <duration>
<def-note-octave> := "o=" <octave>
<def-beats> := "b=" <beats-per-minute>

(c) Revolution Education Ltd.
All rights reserved

```
; If not specified, defaults are
; duration = 4 (quarter note)
; octave = 6
; beats-per-minute = 63 (decimal value)                    PICAXE defaults to 62
<note-command> :=
[<duration>] <note> [<octave>] [<special-duration>] <delimiter>
<duration> :=
”1" |                ; Full 1/1 note
”2" |                ; 1/2 note
”4" |                ; 1/4 note
”8" |                ; 1/8 note
”16" |               ; 1/16 note                  Not used –        PICAXE changes to 8
”32" |               ; 1/32 note                  Not used –        PICAXE changes to 8

<note> :=
”C” |
”C#” |
”D” |
”D#” |
”E” |
”F” |
”F#” |
”G” |
”G#” |
”A” |
”A#” |
”B” |                ; “H” can also be used        PICAXE exports using B
“P” ; pause

<octave> :=
”5" |                 ; Note A is 440Hz
”6" |                ; Note A is 880Hz
”7" |                ; Note A is 1.76 kHz
”8"                  ; Note A is 3.52 kHz          Not used - PICAXE uses octave 7

<special-duration> :=
”.”                  ; Dotted note                 Not used - PICAXE rounds down
<delimiter> := “,”
```


(c) Revolution Education Ltd.
All rights reserved

2.162 uniin

257

Syntaxe:

UNIIN broche, type, commande, (var, var...)

UNIIN broche, type, commande, adresse1, adresse2, (var, var...)

- broche est une variable ou une constante qui précise la broche d'entrée/sortie à utiliser.

- type représente le type de périphérique UNI/O, par exemple %10100000 pour une mémoire EEPROM

- commande est le type de lecture à réaliser, à choisir parmis :
  UNI_READ lit depuis l'adresse indiquée,
  UNI_CRRD lit depuis la position courante
  UNI_RDSR lit l'octet d'état
- adresse1 et adresse2 constituent une adresse optionnelle uniquement utilisée dans le cas UNI_READ
- les variables **var** reçoivent les données.

```
par exemple :
uniin C.3, %10100000, UNI_RDSR, (b1)
uniin C.3, %10100000, UNI_CRRD, (b1,b2,b3)
uniin C.3, %10100000, UNI_READ, 0, 1, (b1,b2,b3)
```

Fonction:
Lit des données depuis un périphérique UNI/O vers des variables PICAXE.

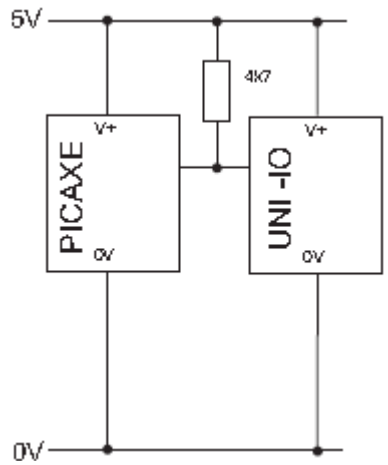
Information:

La commande "uniin" permet de lire des données depuis un périphérique UNI/O, comme par exemple une mémoire EEPROM 11LCxxx. Les puces UNI/O n'ont besoin que d'une seule broche pour dialoguer avec un PICAXE. Une résistance de forçage au niveau haut de 4k7 n'est pas obligatoire au regard des spécification du bus UNI/O, mais reste fortement recommandée.

Cette commande ne peut être utilisée sur la broche suivante en raison de limitations physiques du PIC utilisé :
20X2 C.6 = fixée en entrée

Exemple:

Merci de lire la commande **uniout** *page suivante*



2.163 uniout

258

Syntaxe:

UNIOUT broche, type, commande

UNIOUT broche, type, commande, (donnee)

UNIOUT broche, type, commande adresse1, adresse2, (donnee, donnee...)

- broche est une variable ou une constante qui précise la broche d'entrée/sortie à utiliser.

- type représente le type de périphérique UNI/O, par exemple %10100000 pour une mémoire EEPROM

- commande est le type d'écriture à réaliser, à choisir parmis :
  UNI_WRITE écrit
  UNI_WREN autorise l'écriture
  UNI_WRDI interdit l'écriture
  UNI_WRSR statut d'écriture
  UNI_ERAL efface tout
  UNI_SETAL met tout à un

- adresse1 et adresse2 constituent une adresse optionnelle uniquement utilisée dans le cas UNI_WRITE
- donnee fourni les données à écrire.

par exemple :

```
uniout C.3, %10100000, UNI_ERAL
uniout C.3, %10100000, UNI_SETAL
uniout C.3, %10100000, UNI_WREN
uniout C.3, %10100000, UNI_WRSR, (%0011)
uniout C.3, %10100000, UNI_WRITE, 0, 1, (b1)
uniout C.3, %10100000, UNI_WRDI
```

Fonction:

La commande "uniout" permet d'écrire des données sur un périphérique UNI/O, comme par exemple une mémoire EEPROM 11LCxxx. Les puces UNI/O n'ont besoin que d'une seule broche pour dialoguer avec un PICAXE.

Tenez compte du fait que les mémoires UNI/O utilisent des pages de 16 octets. Une opération d'écriture ne peut franchir la frontière entre deux pages (à savoir un multiple de 16 octets). ceci signifie par exemple :

- que vous pouvez écrire 10 octets avec une seule commande UNI_WRITE à partir de l'adresse 0
- mais que vous ne pourrez le faire à partir de l'adresse 10, puis dans ce cas vous devriez franchir la frontière entre les pages un et deux, frontière située entre les adresses 15 et 16.

Information:

La commande "uniout" permet d'écrire des données sur un périphérique UNI/O, comme par exemple une mémoire EEPROM 11LCxxx. Les puces UNI/O n'ont besoin que d'une seule broche pour dialoguer avec un PICAXE.

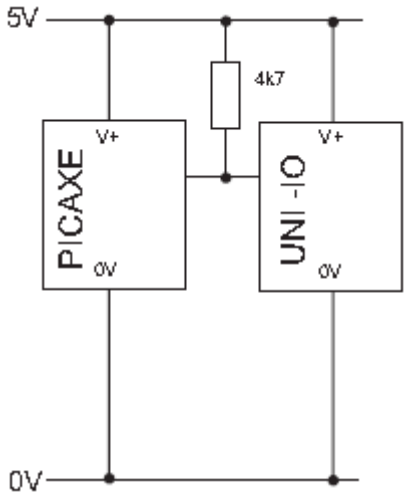
Une résistance de forçage au niveau haut de 4k7 n'est pas obligatoire au regard des spécification du bus UNI/O, mais reste fortement recommandée.

Prenez garde lors de la mise sous tension (ou d'une remise à zéro) au fait que les mémoires UNI/O demeurent dans un état de sommeil en basse consommation. Il est nécessaire de les "réveiller" par une impulsion positive (au moyen de la commande pulsout) avant tout usage des commandes uniin ou uniout.

Cette commande ne peut être utilisée sur la broche suivante en raison de limitations physiques du PIC utilisé :
20X2 C.6 = fixée en entrée

Exemple:

```
reset_uni:
pulsout C.3, 1     ; INDISPENSABLE pour réveiller la mémoire
                   ; au moyen d'une impulsion sur front montant

main:
     inc b1
     uniout C.3, %10100000, UNI_WRSR, (0)          ; efface l'état
     uniout C.3, %10100000, UNI_WREN               ; autorise l'écriture
     uniout C.3, %10100000, UNI_WRITE, 0, 1, (b1)  ; écrit
     pause 10                                      ; attends un peu (pour l'écriture)
     uniout C.3, %10100000, UNI_WRDI               ; interdit l'écriture
     pause 1000                                    ; attends franchement
     uniin C.3, %10100000, UNI_READ, 0, 1, (b2)    ; lit
     debug                                         ; affiche la donnée

     goto main                                     ; boucle
```



(c) Revolution Education Ltd.
All rights reserved

2.164 wait

260

Syntaxe :

WAIT secondes

" secondes " est une constante (1 – 65) qui spécifie la durée de la pause en secondes.

Concerne :

08 / 08M / 08M2

14M / 14M2

18 / 18A / 18M / 18M2 / 18X

20M / 20M2 / 20X2

28A / 28X / 28X1 / 28X2

40X / 40X1 / 40X2

Commentaire :

Entraîne une pause dont la durée est un nombre entier de secondes.

Remarque :

Plutôt réservée aux jeunes utilisateurs, cette instruction est équivalente à " pause X " où X = n * 1000 " (n<= 65).

Elle ne peut-être utilisée avec un variable.

Exemple :

```
main:
      switch on B.7       ; place la sortie B.7 en "1"
            wait 5        ; pause de  5 secondes
      switch off B.7      ; place la sortie B.7 en "0"
            wait 5        ; pause de 5 secondes
goto main                 ; retour au début
```

2.165 write

261

Syntaxe:

WRITE localisation, data, data, WORD wordvariable…

- localisation est une constante ou une variable spécifiant le début de l'enregistrement des données (0 à 255)
- data est une constante ou une variable contenant la donnée à sauvegarder. Le mot clé WORD doit être placé devant les données de type word.

Fonction:

Écrit des données en mémoire EEPROM.

Information:

La commande write sauvegarde une donnée en mémoire EEPROM. Les données sauvegardées en EEPROM sont conservées lorsque le microcontrôleur n'est plus sous tension. La lecture des données se fait par la commande READ.

Pour les PICAXEs 08, 08M, 08M2, 14M, 18, 18M et 18M2, la mémoire EEPROM est partagée avec la mémoire du programme. La taille du programme est donnée par le bouton "syntax check" du menu de PE. Voir la commande EEPROM (DATA) pour plus de détails.

En fait, dans le système PICAXE, une variable de type word est la concaténation de deux variables de type byte ex:w0=(b1,b0).

Il faut donc écrire deux bytes consécutifs pour constituer une variable type word. Avec le mot clé WORD l'enregistrement des bytes se fait dans l'ordre (b0,b1), "poids faible en tête".

Exemple:

```
Debut:
        For b3=0 to 10                  ;début d'une boucle for/next
                Serin C.6,N2400,b4      ;serin reçoit une donnée dans le byte b1
        write b3,b4                     ;la valeur de b4 est enregistrée dans la mémoire b3 de ;l'EEPROM (mémoire 0 au premier passage)
  next b3                               ;boucle pour l'écriture de la mémoire suivant, jusqu'à 10
```

2.166 writemem

262

Syntaxe:

WRITEMEM position,donnee
- position est une variable ou une constante qui Définit une adresse sur un octet (0-255)
- donnee est une variable ou une constante qui donne l'octet à écrire

Fonction:

Ecrit un octet à la position indiquée de la mémoire FLASH.

Information:

La mémoire data des PICAXEs-28A est limitée à 64 octets. En conséquence, la commande **writemem** fourni un espace de stockage suplémentaire de 256 octets dans une zone de mémoire secondaire. Cette dernière n'est pas remise à zéro lors du téléchargement d'un programme.

Cette commande n'est pas disponible sur les PICAXEs-28X puisqu'une mémoire externe I2C peut être utilisée.

La commande **writemem** ne manipule que des octets. Donc, pour écrire une variable word il faut utiliser deux fois cette commande. (Par exemple pour écrire w0, il faut en fait écrire successivement b0 puis b1)

Exemple:

```
main:
        for b0 = 0 to 255               ; boucle 256 fois
                serin 6,N2400,b1        ; réception de données sur un port série
                writemem b0,b1          ; écrit la valeur de b1 à la position b0
        next b0
```


(c) Revolution Education Ltd.
All rights reserved

2.167 writei2c

263

Cette commande est obsolète, pensez à utiliser la commande hi2cout.

Syntaxe:

WRITEI2C location, (variable,…)

WRITEI2C (variable,…)

- variable contient l'octet à écrire.
- location est une variable/constante spécifiant l'octet (ou le word) de départ de l'écriture.

Fonction:

Permet d'envoyer des données sur le bus i2c.

Information:

Cette commande est utilisée pour envoyer des données sur un périphérique i2c.

L'envoi d'une donnée s'effectue à partir de l'adresse de départ indiquée, l'envoi d'autres données séquentielles est possible si le périphérique i2c le supporte.

Une commande i2cslave doit être exécutée avant d'utiliser cette commande.

Exemple:

```
; Utilisation d'un module horloge DS1307
; le PICAXE est maître, le module DS1307 est l'esclave et utilise l'adresse i2c %11010000
; écriture de l'heure à 11:59:00 et de la date au jeudi 17/10/13
; jeudi est le 05° jour de la semaine (dimanche le 01, samedi le 07)
    symbol seconde = b0;
    symbol minute = b1;
    symbol heure = b2;
    symbol jour = b3;
    symbol date = b4;
    symbol mois = b5;
    symbol annee = b6;
    symbol control = b7;
    i2cslave, %11010000, i2cslow, i2cbyte   ; adressage DS1307
                                                    ; mode slow en octet
    let seconde = $00   ; positionne la variable seconde à 00
    let minute = $59    ; positionne la variable minute à 59
    let heure = $11     ; positionne la variable heure à 00
    let jour = $05      ; positionne la variable jour au jeudi donc 05
    let date = $17      ; positionne la variable date à 17
    let mois = $10      ; positionne la variable mois à 10
    let annee = $13     ; positionne la variable annee à 13
    let control = $10   ; clignotement led du module horloge $10
                        ; allumage led du module horloge $80
                        ; extinction led du module horloge $00

    writei2c 0,(seconde,minute,heure,jour,date,mois,annee,control)      ;                                chargement
                                                                        ;des infos dans le module horloge
    end                 ; fin de la mise à l'heure du module horloge
```

3 ANNEXES

3.1 Annexe 1 - Commandes

Les mots suivants représentent des **commandes** et ne peuvent donc être utilisés en tant que noms de variables ou de labels.

adcconfig
backward, bcdtoascii, bcdtobin, bintoascii, bintobcd, booti2c, branch, button
calibadc, calibadc10, calibfreq, call, case, clearbit, compsetup, count
daclevel, dacsetup, data, debug, dec, disablebod, disabletime, disconnect, do,
doze
eeprom, else, elseif, enablebod, enabletime, end, endif, endselect, exit
for, forward, fvrsetup
get, gosub, goto
halt, hi2cin, hi2cout, hi2csetup, hibernate, high, hintsetup, hpwm, hpwmduty,
hpwmout, hserin, hserout, hsersetup, hshin, hshout, hspiin, hspiout, hspisetup
i2cread, i2cslave, i2cwrite, if, inc, infrain, infrain2, infraout, input, inputtype, irin,
irout
kbin, kbled, keyin, keyled
let, lookdown, lookup, loop, low
nap, next
on, output, owin, owout
pause, pauseus, peek, peeksfr, play, poke, pokesfr, pullup, pulsin, pulsout, put,
pwm, pwmduty, pwmout
random, read, readadc, readadc10, readdac, readdac10, readfirmware, readi2c,
readinternaltemp, readmem, readoutputs, readowclk, readowsn, readpinsc,
readportc, readrevision, readsilicon, readtable, readtemp, readtemp12, reconnect,
reset, resetowclk, restart, resume, return, reverse, rfin, rfout, run
select, sensor, serin, serout, serrxd, sertxd, servo, servopos, setbit, setfreq, setint,
setintflags, settimer, shiftin, shiftout, shin, shout, sleep, sound, spiin, spiout,
srlatch, srreset, srset, step, stop, suspend, swap, switch, switchoff, switchon,
symbol
table, tablecopy, tmr3setup, toggle, togglebit, touch, touch16, tune
uniin, uniout, until

wait, while, write, writei2c, writemem

3.2 Annexe 2 - Autres mots-clés réservés

(qui ne sont pas des commandes)

a, a.0-a.7, adcsetup, adcsetup2, and, andnot, atan
b, b.0-b.7, b0-b55, b300_4, b300_8, b300_16, b300_20, b300_32, b300_40,
b300_64, b600_4, b600_8, b600_16, b600_20, b600_32, b600_40, b600_64,
b1200_4, b1200_8, b1200_16, b1200_20, b1200_32, b1200_40, b1200_64,
b2400_4, b2400_8, b2400_16, b2400_20, b2400_32, b2400_40, b2400_64,
b4800_4, b4800_8, b4800_16, b4800_20, b4800_32, b4800_40, b4800_64,
b9600_4, b9600_8, b9600_16, b9600_20, b9600_32, b9600_40, b9600_64,
b14400_4, b14400_8, b14400_16, b14400_20, b14400_32, b14400_40,
b14400_64, b19200_4, b19200_8, b19200_16, b19200_20, b19200_32,
b19200_40, b19200_64, b28800_4, b28800_8, b28800_16, b28800_20,
b28800_32, b28800_40, b28800_64, b31250_4, b31250_8, b31250_16,
b31250_20, b31250_32, b31250_40, b31250_64, b38400_4, b38400_8,
b38400_16, b38400_20, b38400_32, b38400_40, b38400_64, b57600_4,
b57600_8, b57600_16, b57600_20, b57600_32, b57600_40, b57600_64,
b76800_4, b76800_8, b76800_16, b76800_20, b76800_32, b76800_40,
b76800_64, b115200_4, b115200_8, b115200_16, b115200_20, b115200_32,
b115200_40, b115200_64, bit, bit0-bit31, bptr, bptr0-bptr7, @bptr, @bptrdec,
@bptrinc
c, c.0-c.7, clear, cls, compflag, compvalue, cos, cr
d, d.0-d.7, dcd, dig, dir0-dir7, dira.0-dira.7, dirb.0-dirb.7, dirc.0-dirc.7, dird.0-dird.7, dirs, dirsa, dirsb, dirsc, dirsd
em4, em8, em16, em20, em32, em40, em64
flag0-flag15, flags, flagsh, flagsl, fvr1024, fvr2048, fvr4096
hi2cflag, hi2clast, hint0flag, hint1flag, hint2flag, hintflag, hserflag, hserinflag, hserinptr, hserptr

i2cbyte, i2cfast, i2cfast_4, i2cfast_8, i2cfast_16, i2cfast_20, i2cfast_32, i2cfast_40,
i2cfast_64, i2cfast4, i2cfast8, i2cfast16, i2cfast20, i2cfast32, i2cfast40, i2cfast64,
i2cmaster, i2cslow, i2cslow_4, i2cslow_8, i2cslow_16, i2cslow_20, i2cslow_32,
i2cslow_40, i2cslow_64, i2cslow4, i2cslow8, i2cslow16, i2cslow20, i2cslow32,
i2cslow40, i2cslow64, i2cword, infra, input0-input7, inv, is, it_5v0, it_4v5, it_4v0, it_3v5, it_3v3, ir_3v0, ir_raw_h, it_raw_l
k31, k62, k125, k250, k500, keyvalue
lf, lsbfirst, lsbfirst_h, lsbfirst_l, lsbpost, lsbpost_h, lsbpost_l, lsbpre, lsbpre_h, lsbpre_l
m1, m2, m4, m8, m16, m32, m64, max, min, mod, msbfirst, msbfirst_h,
msbfirst_l, msbpost, msbpost_h, msbpost_l, msbpre, msbpre_h, msbpre_l
n300, n300_4, n600, n600_4, n600_8, n1200, n1200_4, n1200_8, n2400,
n2400_4, n2400_8, n2400_16, n4800, n4800_4, n4800_8, n4800_16,
n4800_32, n9600, n9600_8, n9600_16, n9600_32, n9600_64, n19200,
n19200_16, n19200_32, n19200_64, n38400, n38400_32, n38400_64, n76800,
n76800_64, nand, ncd, nob, nor, not
off, or, ornot, outpin0-outpin7, outpina.0-outpina.7, outpinb.0-outpinb.7,
outpinc.0-outpinc.7, outpind.0-outpind.7, outpins, outpinsa, outpinsb, outpinsc,
outpinsd, output0-output7, ownoreset, ownoreset_bit, owresetafter,
owresetafter_bit, owresetbefore, owresetbefore_bit, owresetboth,
owresetboth_bit, owresetfirst, owresetfirst_bit
pin0-pin7, pina.0-pina.7, pinb.0-pinb.7, pinc.0-pinc.7, pind.0-pind.7, pins,
pinsa, pinsb, pinsc, pinsd, port, porta, portb, portc, portd, pot, ptr, ptr0-ptr15,
ptrh, ptrl, @ptr, @ptrdec, @ptrincpwmdiv16, pwmdiv4, pwmdiv64, pwmfull_f, pwmfull_r, pwmhalf, pwmhhhh, pwmhlhl, pwmlhlh,
pwmllll, pwmsingle

s_w0-s_w7, sensor, set, sin, spifast, spimedium, spimode00, spimode00e, spimode01, spimode01e, spimode10, spimode10e,
spimode11, spimode11e, spislow, sqr, step

t300, t300_4, t600, t600_4, t600_8, t1200, t1200_4, t1200_8, t2400, t2400_4, t2400_8, t2400_16, t4800, t4800_4, t4800_8,
t4800_16, t4800_32, t9600, t9600_8, t9600_16, t9600_32, t9600_64, t19200, t19200_16, t19200_32, t19200_64, t38400, t38400_32,
t38400_64, t76800, t76800_64, t1s_4, t1s_8, t1s_16, t1s_20, t1s_32, t1s_40, t1s_64, task, then, time, timer, timer3, to, toflag, trisc

uni_crrd, uni_eral, uni_rdsr, uni_read, uni_setal, uni_wrdi, uni_wren, uni_write, uni_wrsr, until

w0-w27, while, word

xnor, xor, xornot

(c) Revolution Education Ltd.
All rights reserved

3.3 Annexe 3 : Labels réservés

Les étiquettes suivantes ont une signification spéciale et ne doivent être utilisées que dans un contexte bien précis :

interrupt:
gestion des interruptions : voyez la commande **setint**

start0:
start1:
start2:
start3:
start4:
start5:
start6:
start7:

traitements en parallèle : voyez la commande **restart**

(c) Revolution Education Ltd.
All rights reserved

3.4 Annexe 4 - Commandes potentiellement en conflit

Tâches fondées sur des interruptions internes :

Tâche	Interruption interne	Commande
Background serial receive	Serial interrupt	**hsersetup**
Background I2C slave mode	I2C interrupt	**hi2csetup**
Timer	Timer 1 interrupt	**settimer**
Servo	Timer 1 & 2 interrupts	**servo**
Timer 3	Timer 3 interrupt	**tmr3setup**
Hardware pin interrupt	Hardware pin interrupt	**hintsetup**
Comparator	Comparator interrupt	**compsetup**

Les fonctions citées ci-dessus utilisent des tâches conçues à partir des interruptions internes du microcontrôleur.

Les interruptions internes arrêtent temporairement le déroulement du programme principal pour effectuer le travail demandé par l'interruption. L'utilisateur final ne se rends normalement pas compte de cet arrêt momentané dans la mesure ou le traitement effectué automatiquement est très rapide.

Néanmoins, ce procédé peut poser des problèmes pour les commandes sensibles à la temporisation comme les communications série et One-Wire. Si une interruption se produit juste au moment où le PICAXE traite une commande sensible, alors il est possible que la temporisation soit erronée et que des données soient corrompues aussi bien en entrée qu'en sortie du PICAXE. En conséquence, les commandes suivantes doivent temporairement désactiver toutes les interruptions lors de leur exécution :

port série RS232	**serin, serout, serrxd, sertxd, debug**
One Wire	**owin, owout, readtemp, readtemp12, readowsn**
UNI/O	**uniin, uniout**
Infra -red	**infraout, irout**

Veuillez noter que les autres commandes utilisant le temps (par exemple **count, pulsin, pulsout**, etc...) ne désactivent pas les interruptions, mais, si certaines sont actives, le temps de traitement des interruptions matérielles risque de dégrader la précision de ces commandes lors de leur exécution.

Le programme principal doit en tenir compte et contourner ces limitation du microcontrôleur.

NDT : ces limitations rendent en pratique la réception en tâche de fond (background receive) inutilisables pour réceptionner à la volée des données et les retransmettre sur un autre port. Par exemple si vous lisez des données sur un port série et que vous souhaitez les retransmettre vers un afficheur série, il faudra établir un protocole, sans quoi des données seront perdues ou erronées.

Tâches de fond dépendant de la fréquence du PICAXE

Tâche	Module interne	Commande
PWM	Timer 2 & pwm	**pwmout / hpwm**
Background serial receive	Serial receive	**hsersetup**
Background I2C slave mode	I2C receive	**hi2csetup**
Servo	Timer 1 & 2	**servo**
Timer	Timer 1	**settimer**
Timer 3	Timer 3	**tmr3setup**

Veuillez noter que ces tâches dépendent de la fréquence du microprocesseur. Ceci a deux conséquences principales :

1) La commande **servo** ne peut être utilisée en même temps que les commandes **pwm/hpwm/timer** dans la mesure où elles utilisent également les timers 1 & 2.

2) Certaines commandes M2, X1 ou X2 comme "**readtemp**" basculent automatiquement et temporairement sur le résonateur interne à 4 MHz pour effectuer leur travail (de façon à être certain que les opérations sensibles au temps soient réalisées correctement). Quand ceci se produit, les travaux réalisés en tâche de fond peuvent être affectés. Par exemple une onde générée par la commande **pwmout** peut temporairement basculer sur la fréquence correspondant à 4MHz.

(c) Revolution Education Ltd.
All rights reserved

3.5 Annexe 5 - variantes du X2

La plupart des commandes X2 sont implémentées sur toute la gamme des PICAXEs X2. Néanmoins, les différentes variantes de PICAXEs-X2 disposent de possibilités et de tailles mémoires assez différentes. Ceci est du aux caractéristiques variables des PICs utilisés pour fabriquer les PICAXEs. Il n'est pas possible pour le microcode du PICAXE de compenser ces différences dans la mesure où il s'agit de fonctionnalités implémentées physiquement sur le silicone de chaque puce.

Fonctionnalité	***commande PICAXE***	***20X2***	***28X2***	***28X2*** -5V	***28X2*** -3V	***40X2***	***40X2*** -5V	***40X2*** -3V
PIC de base (série PIC18F...)		14K22	25K22	2520	25K20	45K22	4520	45K20
Plage de tension (V)		1.8-5.5	2.1-5.5	4.5-5.5	1.8-3.6	2.1-5.5	4.5-5.5	1.8-3.6
plage de Version du microcode du PICAXE		C.0+	B.3+	B.0-B.2	B.A-B.C	B.3+	B.0-B.2	B.A-B.C
Toujours en production		Oui	Oui	Non	Non	Oui	Non	Non
Fréquence maxi MHz	setfreq	64	16	8	16	16	8	16
Fréquence mini Mhz	setfreq	n/a	64	40*	64	64	40*	64
Support des touches sensitives	touch	Non	Oui	Non	Non	Oui	Non	Non
Paramétrage de l'ADC en séquence ou individuel	adcsetup	ind.	ind.	seq.	ind.	ind.	seq.	ind
tension de référence interne de l'ADC (V)	calibadc	1.024	1.024	Non	1.2	1.024	Non	1.2
Variables RAM (octets)	peek, poke, @bptr	128	256	256	256	256	256	256
Mémoire tampon (octets)	put, get, @ptr	128	1024	1024	1024	1024	1024	1024
Nonmbre de logements de programmes internes	run	1	4	4	4	4	4	4
Nonmbre de logements de programmes externes	run	32	32	32	32	32	32	32
Broches d'nterruption matérielles	hintsetup	2	3	3	3	3	3	3
Sorties PWM	pwmout	1	4	2	2	2	2	2
Support du PWM	hpwm	Oui	Oui	Non	Oui	Oui	Oui	Oui
Frein en mode PWM	hpwm	Oui	Oui	Non	Oui	Oui	Non	Oui
Résistances vers l'alimentation	pullup	Oui	Oui	Non	Oui	Oui	Non	Oui
Srlatch, FVR et module DAC	srlatch, fvrsetup, dscsetup	Oui	Oui	Non	Non	Oui	Non	Non

* 32MHz (résonateur 8MHz avec PLL x4) est recommandée pour les programmes utilisant le port série dans la mesure où 40 Mhz n'étant pas un multiple pair de 8, il n'est en conséquence pas possible de générer des vitesses de transmission valides.

(c) Revolution Education Ltd.
All rights reserved

3.6 Annexe 6 - variantes du M2

La plupart des commandes M2 sont supportées par tous les PICAXEs M2.

Néanmoins, certaines variantes de PICAXEs M2 disposent de fonctionnalités et de tailles mémoires assez différentes, et résumées dans le tableau ci-dessous. Ceci est du aux caractéritiques physiques des PICs utilisés pour construire chaque PICAXE. Il est impossible d'implémenter ces fonctionnalités dans le micro-logiciel embarqué dans chaque PICAXE dans la mesure où il s'agit de fonctionnalités gravées dans le silicone.

Fonctionnalité	Commande PICAXE	08M2	18M2	18M2+	14M2	20M2
Plage de tension (V)		2,3-5,5	1,8-5,5	1,8-5,5	1,8-5,5	1,8-5,5
Capacité mémoire (Octets)		2048	2048	2048	2048	2048
Tâches parallèles (starts)	resume, suspend	4	4	8	8	8
Fréquence interne maxi (MHz)	setfreq	32	32	32	32	32
Variables RAM (octets)	peek, poke	128	256	512	512	512
Table data (octets)	table, readtable, tablecopy	-	-	512	512	512
I2C master supporté	hi2cin, hi2cout, hi2csetup	Oui	Oui	Oui	Oui	Oui
Canaux PWM	pwmout	1	2	2	4	4
Hpwm supporté	hpwm	Non	Non	Non	Oui	Oui
Clavier supporté	kbin, kbled	Non	Non	Oui	Oui	Oui
RF radio supporté	rfin, rfout	Non	Non	Oui	Oui	Oui
Capteur interne de température	readinternaltemp	Oui	Non	Oui	Oui	Oui
Type d'entrée configurable	inputtype	Non	Non	Non	Oui	Oui

(c) Revolution Education Ltd.
All rights reserved

3.7 Site WEB du fabricant:

Site web principal : www.PICAXE.com
Forum: www.PICAXEforum.co.uk
VSM Simulator: www.PICAXEvsm.com

Les produits PICAXE sont conçus et distribués par :

Revolution Education Ltd
http://www.rev-ed.co.uk/

(c) Revolution Education Ltd.
All rights reserved

3.8 Marque déposée :

PICAXE® est une marque déposée appartenant à Microchip Technology Inc.

Revolution Education n'est pas un agent ou un représentant de Microchip et n'a aucune autorité pour représenter Microchip de quelque façon que ce soit.

(c) Revolution Education Ltd.
All rights reserved

3.9 Remerciements:

Revolution Education remercie vivement :

Clive Seager
John Bown
LTScotland
Higher Still Development Unit
UKOOA
Mike Meakin de Nikam Electronics qui a gentiment mis à disposition le code interne du NKM2401 utilisé pour les commandes rfin et rfout ainsi que pour le kit AXE213.

(c) Revolution Education Ltd.
All rights reserved