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.

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
```


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°

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 !

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.