Návěští

Slouží k označení míst v programu pro příkazy GOTO, GOSUB, BRANCH. Musí začínat písmenem nebo podtržítkem, končit dvojtečkou. Návěští nesmí být shodné s rezervovaným klíčovým slovem (příkazem) a nesmí obsahovat znaky s diakritikou.

Například:

```
main:       high 1              ' switch on output 1
            pause 5000          ' wait 5 seconds
            low 1               ' switch off output 1
            pause 5000          ' wait 5 seconds
            goto main           ' loop back to start
```

Mezery (whitespace)

Mezery, tabulátory a oddělovače řádků (enter) mohou být použity kdekoliv v programu – například pro zvýšení čitelnosti.

Komentáře

Komentáře jsou uvozeny apostrofem ('), středníkem nebo klíčovým slovem REM a platí do konce řádku.

Například:

```
high 0              ' make output 0 high
high 0              ; make output 0 high
high 0             REM make output 0 high
```

Konstanty

Desítkové (dekadické)	– číslo bez dalšího označení
Dvojkové (binární)	– s prefixem %, například %00101100
Šestnáctkové (hexadecimální)	– s prefixem $, například $F5
Rozsah 16 bitů	– 0 až 65535

Například:

```
100                 ' 100 decimal
$64                 ' 64 hex
%01100100           ' 01100100 binary
"A"                 ' "A" ascii (65)
"Hello"             ' "Hello" - equivalent to "H","e","l","l","o"
B1 = B0 ^ $AA       ' xor variable B0 with AA hex
```

Symboly

Konstantám a proměnným mohou být přiřazeny symbolická jména, což přispívá k čitelnosti programu. Tato jména nesmí obsahovat znaky s diakritikou.

Například:

```
symbol RED_LED = 7      ' define a constant symbol
symbol COUNTER = b0     ' define a variable symbol
let COUNTER = 200       ' preload variable with value 200
main:                   ' define a program address
                        ' address symbol end with colons
high RED_LED            ' switch on output 7
pause COUNTER           ' wait 0,2 seconds
low RED_LED             ' switch off output 7
pause COUNTER           ' wait 0,2 seconds
goto main               ' loop back to start
```

Proměnné

Univerzální proměnné

K dispozici je 14 byte b0 až b13, které mohou být sdružovány po dvou v 16 bitová slova (word) w0 až w6. Jednotlivé bity b0 a b1 mohou být také adresovány jako bit0 až bit 15.

Všechny univerzální proměnné jsou bez znaménka (unsigned) a na začátku běhu programu obsahují hodnotu nula.

word	***vyšší byte***	***nižší byte***
w0	b1	b0
w1	b3	b2
w2	b5	b4
w3	b7	b6
w4	b9	b8
w5	b11	b10
w6	b13	b12

byte	***MSB***							***LSB***
b0	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
b1	bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8

Speciální proměnné

pins – při čtení hodnoty (tj. napravo od rovnítka) odpovídá vstupům, při přiřazení hodnoty (tj. nalevo od rovnítka) odpovídá výstupům. Pro testování hodnoty jednotlivých vstupů v příkazu IF se dělí na jednotlivé bity. Bitové proměnné jsou přiřazené pouze existujícím vstupům.

byte	***MSB***						***LSB***
pins	pin7	pin6	–	–	–	pin2	pin1

infra – speciální proměnná pro použití v příkazu INFRAIN

keyvalue – speciální proměnná pro použití v příkazu KEYIN, překrývá se s proměnnou infra

Operátory

Aritmetické

+	sčítání
–	odčítání
*****	násobení
******	při násobení 16 x 16 bitů předá vyšších 16 bitů výsledku
/	dělení
//	zbytek po dělení (modulo)
%	zbytek po dělení (modulo) – alternativní zápis
MAX	omezí výsledek shora – na maximální hodnotu
MIN	omezí výsledek zdola – na minimální hodnotu

Logické – fungují po jednotlivých bitech

|	AND	– logický součin
&	OR	– logický součet
^	XOR	– výlučně nebo
&/	AND NOT	– logický součin s druhým operandem negovaným, toto není NAND
|/	OR NOT	– logický součet s druhým operandem negovaným, toto není NOR
^/	XNOR	– logická ekvivalence

Výrazy

Přípustné jsou pouze jednoduché výrazy bez závorek a přednosti operátorů, vyhodnocuje se zleva doprava.

Například:
b0 = b1 + 12 / b3 což odpovídá obvyklejšímu zápisu b0 = (b1 + 12) / b3
w1 = 3 * w0 max 445

Abecední seznam příkazů

bcdtoascii

syntaxe:
BCDTOASCII proměnná, desítky, jednotky
BCDTOASCII proměnná_word, tisíce, stovky, desítky, jednotky
- proměnná obsahuje hodnotu 0-99 (bajt) nebo 0-9999 (slovo)
- tisíce ascii hodnota („0“ až „9“)
- stovky ascii hodnota („0“ až „9“)
- desítky ascii hodnota („0“ až „9“)
- jednotky ascii hodnota („0“ až „9“)

funkce:
Konvertuje hodnotu BCD na samostatné bajty ASCII.

Tento pseudopříkaz je určený ke zjednodušení konverze bajtů nebo slov hodnot BCD na ASCII. Pamatujte si, že maximání platná hodnota BCD je 99 (bajt) nebo 9999 (slovo).

Například:
```
main:       inc b1
            bcdtoascii b1,b2,b3     ' konverze na ascii
            debug                   ' ladění pro otestování
            goto main               ' zpátky na začátek
```

bintoascii

syntaxe:
BINTOASCII proměnná, stovky, desítky, jednotky
BINTOASCII proměnná_slovo, desítky_tisíc, tisíce, stovky, desítky, jednotky
- proměnná obsahuje hodnotu 0-255 (bajt) nebo 0-65535 (slovo)
- desítky_tisíc ascii hodnota („0“ až „9“)
- tisíce ascii hodnota („0“ až „9“)
- stovky ascii hodnota („0“ až „9“)
- desítky ascii hodnota („0“ až „9“)
- jednotky ascii hodnota („0“ až „9“)

funkce:
Konvertuje binární hodnotu na samostatné bajty ASCII.

Tento pseudopříkaz je určený ke zjednodušení konverze binárních hodnot bajtu nebo slova na ASCII.

Například:
```
main:       inc b1
            bintoascii b1,b2,b3,b4  ' konverze b1 na ascii
            debug                   ' ladění pro otestování
            goto main               ' zpátky na začátek
```

branch

syntaxe:
BRANCH offset,(address0,address1…addressN)
- offset je proměnná, určující na kterou adresu (0-N) se má skočit.
- adresy (address) jsou návěští, na která se větví program podle hodnoty proměnné offset.

funkce:

Tento příkaz umožňuje větvení programu podle proměnné offset. Pokud je její hodnota 0, skočí se na první návěští, pokud je hodnota 1, skočí se na druhé atd. Pokud je hodnota větší nežli odpovídá poslednímu uvedenému návěští, neprovede se žádný skok a program pokračuje na dalším řádku.

Například:

```
reset:     let b1 = 0
           low 0
           low 1
           low 2
           low 3
main:      let b1 = b1 + 1
           if b1 > 3 then reset
           branch b1,(btn0, btn1, btn2, btn3)
btn0:      high 0
           goto main
btn1:      high 1
           goto main
btn2:      high 2
           goto main
btn3:      high 3
           goto main
```

button

syntaxe:

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

funkce:

Příkaz ke čtení tlačítka, odstranění zákmitů a simulaci opakovaného stisku (autorepeat). Aby správně fungoval, musí být periodicky vykonáván.

Podrobněji je příkaz vysvětlen v anglické příručce.

calibfreq

syntaxe:

CALIBFREQ {-} factor

- factor je konstanta nebo proměnná, nabývající hodnot -31 až 31

funkce:

Slouží k jemnému doladění vnitřního oscilátoru PICAXE. Po připojení napájení má konstanta hodnotu 0. Kladné hodnoty kmitočet oscilátoru zvyšují, záporné hodnoty ho snižují.

count

syntaxe:

COUNT pin, period, variable

- pin je proměnná nebo konstanta, určující, na kterém vstupu se počítají impulsy.

- period je proměnná nebo konstanta, určující dobu měření (1-65535 ms při frekvenci oscilátoru 4 MHz).

- variable je proměnná, do které se zaznamená výsledek (přednostně proměnná word – s rozsahem 0-65535).

funkce:

Slouží k čítání impulsů na vstupu, počítá vzestupné hrany na určeném vstupu. Nejvyšší měřitelná frekvence vstupního signálu je 25 kHz (pokud je střída signálu 1:1) pro frekvenci oscilátoru 4 MHz.

Například:

```
main:      count 1, 5000, w1        ' count pulses in 5 seconds
           debug w1                 ' display value
           goto main                ' else loop back to start
```

debug

syntaxe:

DEBUG {var}

- var je nepovinná proměnná, uvedená pouze z důvodu zpětné programové kompatibility, její hodnota nemá žádný vliv.

funkce:

Zobrazí hodnoty všech proměnných v ladícím okně na monitoru PC. Musí být připojen komunikační kabel. Vzhledem k množství přenášených dat tento příkaz značně zpomaluje běh programu. Rychlejší selektivní výpis ladících informací umožňuje příkaz SERTXD.

Například:

```
main:      let b1 = b1 + 1    ' increment value of b1
           readadc 2,b2       ' read an analogue value
           debug b1           ' display values on computer screen
           pause 500          ' wait 0.5 seconds
           goto main          ' loop back to start
```

dec

syntaxe:

DEC var

- var is the variable to decrement

funkce:

Dekrementace hodnoty proměnné (snížení o 1).

Tento příkaz je zkráceným zápisem „let var = var - 1“

Například:

```
for b1 = 1 to 5
dec b2
next b1
```

disablebod

syntaxe:

DISABLEBOD

funkce:

Vypnutí funkce detekce podpětí.

Některé mikroprocesory PICAXE mají programovatelnou interní funkci detekce podpětí pro automatické resetování čipu při zjištění podpětí. Detekce podpětí je při spuštění programu vždy implicitně zapnuta. Někdy je samozřejmě lepší tuto funkci vypnout pro omezení odběru proudu v aplikacích napájených z baterií, když je čip ve stavu spánku.

Použijete-li příkaz disablebod před příkazem sleep, značně se sníží odběr proudu při provádění příkazu sleep.

Například:

```
main:      disablebod   ' vypnutí detekce podpětí
           sleep 10     ' uspání na 23 sekund
           enablebod    ' zapnutí detekce podpětí
           goto main    ' zpátky na začátek
```

do…loop

syntaxe:
DO
{kód}
LOOP UNTIL/WHILE VAR ?? COND
DO
{kód}
LOOP UNTIL/WHILE VAR ?? COND AND/OR VAR ?? COND…
DO UNTIL/WHILE VAR ?? COND
{kód}
LOOP
DO UNTIL/WHILE VAR ?? COND AND/OR VAR ?? COND…
{kód}
LOOP

- **proměnná** testovaná proměnná
- **podmínka** podmínka pro opuštění cyklu
- **??** může být některá z následujících podmínek
  - **=** rovná se
  - **is** rovná se (alternativní zápis)
  - **<>** nerovná se
  - **!=** nerovná se (alternativní zápis)
  - **>** větší než
  - **>=** větší nebo rovno
  - **<** menší než
  - **<=** menší nebo rovno

funkce:
Cyklus je prováděn, dokud je podmínka splněna (příkaz while) nebo nesplněna (příkaz until)

Tato struktura tvoří cyklus umožňující opakování při splnění nebo nesplnění dané podmínky. Podmínka může být uvedena na řádku s „do“ (splnění podmínky je kontrolováno před provedením kódu) nebo na řádku s „loop“ (splnění podmínky je kontrolováno až po provedení kódu).
K předčasnému opuštění cyklu do…loop je možno použít příkaz exit.

Například:

```
do
high 1
pause 1000
low 1
pause 1000
inc b1
if pin1 = 1 then exit
loop while b1 < 5
```

data
eeprom

syntaxe:

DATA {location},(data,data…)

EEPROM {location},(data,data…)

- location je nepovinná konstanta (0-255), určující počáteční adresu paměti eeprom, kam se budou ukládat data. Pokud není počáteční adresa uvedena, ukládání pokračuje tam, kde předchozí příkaz skončil. Při prvním použití příkazu se začíná na adrese 0.

- data jsou konstanty (0-255), které budou uloženy v paměti eeprom.

funkce:

Příkazy DATA a EEPROM jsou synonyma, oba slouží k naplnění paměti eeprom konstantami při zavádění programu z PC. Tyto konstanty lze načíst v programu příkazem READ. Příkaz DATA či EEPROM neovlivňuje délku programu.

U PICAXE-08, 08M a 18 je EEPROM sdílena s pamětí programu. Pouze nevyužitá paměť může být zužitkována. Délku vlastního programu lze zjistit pomocí příkazu ‚Check Syntax' z menu PICAXE.

Dostupné adresy jsou pak:

PICAXE-08: 0 až (127 - délka programu)

PICAXE-08M: 0 až (255 - délka programu)

PICAXE-18: 0 až (127 - délka programu)

U následujících kontrolérů je paměť eeprom zcela oddělená, takže nemůže dojít ke konfliktu:

PICAXE-28, 28A: 0 až 63

PICAXE-28X, 40X: 0 až 127

PICAXE-18A, 18X: 0 až 255

Například:

```
            EEPROM 0,("Hello World")      ' save values in EEPROM
main:       for b0 = 0 to 10              ' start a loop
            read b0,b1                    ' read value from EEPROM
            serout 7,N2400,(b1)           ' transmit to serial LCD module
            next b0                       ' next character
```

enablebod

syntaxe:

ENABLEBOD

funkce:

Zapnutí detekce podpětí.

Některé jednočipy PICAXE mají programovatelnou interní funkci detekce podpětí pro automatické resetování čipu při zjištění podpětí. Detekce podpětí je při spuštění programu vždy implicitně zapnuta. Někdy je samozřejmě lepší tuto funkci vypnout pro omezení odběru proudu v aplikacích napájených z baterií, když je čip ve stavu spánku.

Použijete-li příkaz disablebod před příkazem sleep, značně se sníží odběr proudu při provádění příkazu sleep.

Například:

```
main:       disablebod  ' vypnutí detekce podpětí
            sleep 10    ' uspání na 23 sekund
            enablebod   ' zapnutí detekce podpětí
            goto main   ' zpátky na začátek
```

end

syntaxe:
END

funkce:
Ukončí běh programu a uvede kontroler do režimu s minimální spotřebou. Obnova běhu programu je možná pouze vypnutím napájení, přivedením nízké úrovně na vstup MCLR (resetem) nebo zavedením nového programu z PC.

Příkaz END vypíná všechny časovače, po jeho provedení se ukončí činnost příkazů PWM a SERVO a kontrolér přejde do režimu s nízkým příkonem. Pokud je tento efekt nežádoucí, lze použít příkaz STOP**.**

Například:

```
main:     let b2 = 15             ' set b2 value
          pause 2000              ' wait for 2 seconds
          gosub flsh              ' call sub-procedure
          let b2 = 5              ' set b2 value
          pause 2000              ' wait for 2 seconds
          end                     ' stop accidentally falling into sub
flsh:     for b0 = 1 to b2        ' define loop for b2 times
          high 1                  ' switch on output 1
          pause 500               ' wait 0,5 seconds
          low 1                   ' switch off output 1
          pause 500               ' wait 0,5 seconds
          next b0                 ' end of loop
          return                  ' return from sub-procedure
```

exit

syntaxe:
EXIT

funkce:
Příkaz exit se používá k okamžitému ukončení cyklu do…loop nebo for…next.

Příkaz exit okamžitě ukončí cyklus do…loop nebo for…next.
Je ekvivalentem příkazu goto pro skok na řádek za koncem cyklu.

Například:

```
main:     do                ' začátek cyklu
          if b1 = 1 then
          exit
          end if
          loop              ' opakování
```

for…next

syntaxe:
FOR variable = start TO end {STEP {-}increment}
..
příkazy programové smyčky
..
NEXT {variable}
- variable je proměnná, která je použita jako čítač cyklů
- start je počáteční hodnota čítače

- end je konečná hodnota čítače

- increment je nepovinná hodnota kroku čítače. Pokud není uvedena použije se hodnota +1. Pokud je uvedena záporna hodnota, předpokládá se, že Start je větší nežli End a čítá se směrem dolů.

funkce:

Slouží k opakovanému provádění kódu uvedeného mezi příkazy FOR a NEXT. Při použití proměnné byte je největší možný počet cyklů 255. Při každém provedení příkazu NEXT se hodnota čítače zvětší (nebo zmenší) o předepsaný krok a porovná se s konečnou hodnotou End. Pokud je čítač větší (nebo menší při záporném kroku) nežli End, smyčka se ukončí a program pokračuje dalším řádkem za NEXT. Příkaz FOR-NEXT může mít osm úrovní vnoření.

Například:

```
main:       for b0 = 1 to 20  ' define loop for 20 times
            high 1            ' switch on output 1
            pause 500         ' wait 0.5 seconds
            low 1             ' switch off output 1
            pause 500         ' wait 0.5 seconds
            next b0           ' end of loop
            pause 2000        ' wait for 2 seconds
            goto main         ' loop back to start
```

gosub

syntaxe:

GOSUB address

- address je návěští podprogramu, který příkaz GOSUB volá

funkce:

Předává řízení programu na udanou adresu, po vykonání příkazu RETURN se vrátí vykonávání programu na řádek následující po příkazu GOSUB. Příkaz GOSUB se liší od příkazu GOTO v tom, že uchovává návratovou adresu. Po každém příkazu GOSUB musí následovat vykonání příkazu RETURN, jinak by došlo k přeplnění zásobníku návratových adres. Příkazy GOSUB mohou mít čtyři úrovně vnoření.

Program kontrolérů 18X, 28X a 40X může obsahovat celkem 15 nebo 255 příkazů GOSUB, podle nastavení v menu Options. U ostatních kontrolérů je povoleno 15, případně 16 příkazů GOSUB v celém programu.

Například:

```
main:       let b2 = 15       ' set b2 value
            pause 2000        ' wait for 2 seconds
            gosub flsh        ' call sub-procedure
            let b2 = 5        ' set b2 value
            pause 2000        ' wait for 2 seconds
            gosub flsh        ' call sub-procedure
            end               ' stop accidentally falling into sub
flsh:       for b0 = 1 to b2  ' define loop for b2 times
            high 1            ' switch on output 1
            pause 500         ' wait 0.5 seconds
            low 1             ' switch off output 1
            pause 500         ' wait 0.5 seconds
            next b0           ' end of loop
            return            ' return from sub-procedure
```

goto

syntaxe:

GOTO address

- address je návěští, na které se předá provádění programu – nepodmíněný skok na jiné místo v programu.

Například:

```
main:       high 1              ' switch on output 1
            pause 5000          ' wait 5 seconds
            low 1               ' switch off output 1
            pause 5000          ' wait 5 seconds
            goto main           ' loop back to start
```

high

syntaxe:

HIGH pin

- pin je proměnná nebo konstanta, označuje výstup, který se použije

funkce:

Nastaví vysokou výstupní úroveň. (U PICAXE-08 zároveň nastaví vývod jako výstupní.)

Například:

```
main:       high 1              ' switch on output 1
            pause 5000          ' wait 5 seconds
            low 1               ' switch off output 1
            pause 5000          ' wait 5 seconds
            goto main           ' loop back to start
```

high portc **(použitelné pro PICAXE 14M)**

syntaxe:

HIGH PORTC pin

- pin je proměnná nebo konstanta, označuje výstup na portu C, který se použije

funkce:

Nastaví vysokou výstupní úroveň na výstupu portu C.

Například:

```
main:       high portc 1        ' switch on output 1
            pause 5000          ' wait 5 seconds
            low portc 1         ' switch off output 1
            pause 5000          ' wait 5 seconds
            goto main           ' loop back to start
```

hpwm **(použitelné pro PICAXE 14M)**

syntaxe:

HPWM režim, polarita, nastavení, čas, prodleva

HPWM OFF

- režim je proměnná/konstanta udávající hardwarový režim PWM

pwmsingle - 0

pwmhalf - 1

pwmfull_f - 2

pwmfull_r - 3

- **polarita** je proměnná/konstanta udávající aktivní polaritu (DCBA)
  pwmHHHH - 0
  pwmLHLH - 1
  pwmHLHL - 2
  pwmLLLL - 3
- **nastavení** je proměnná/konstanta udávající specifické nastavení režimu single - bitová maska %0000 až %1111 pro zapnutí nebo vypnutí DCBA
  režim half – doba mrtvé zóny (hodnota 0-127)
  režim full – nepoužívá se, zadejte 0 jako implicitní hodnotu
- **čas** je proměnná/konstanta (0-255) nastavující dobu PWM
  (doba je délka jednoho cylu zapnutí/vypnutí, tj. celkový čas mark:space).
- **prodleva** je proměnná/konstanta (0-1023) nastavující pracovní cyklus PWM.

funkce:

Hardwarová PWM je metodou řízení motoru pomocí PWM. Může využívat mnoho výstupů a režimů definovaných interním hardwarem PWM mikrokontroléru PIC.

Příkaz **hpwm** může být používán namísto příkazu **pwmout**, ale nikoli současně. Příkaz **pwmout** však může být používán také.

hpwm umožňuje přísutp k řadiči PWM mikrokontroléru PIC. Využívá až 4 piny, které jsou zde pro přehlednost označeny A,B,C,D. Některé z těchto pinů jsou implicitně nastaveny jako vstupy, v takovém případě budou při zpracování příkazu hpwm automaticky přepnuty na výstupy.

A je vstup 2 (C.5)

B je vstup 1 (C.4)

C je vstup 0 (C.3)

D je výstup 5 (C.2)

Ne všechny piny se používají ve všech režimech **hpwm**. Nevyužité bity slouží jako normální vstupně výstupní piny.

single – A a/nebo B a/nebo C a/nebo D (každý bit je možno zvolit)

half – pouze A, B

full – A, B, C, D

Aktivní polarita každé dvojice pinů může být vybrána nastavením polarity:

pwm_HHHH - A a C aktivní v H, B a D aktivní v H

pwm_LHLH - A a C aktivní v H, B a D aktivní v L

pwm_HLHL - A a C aktivní v L, B a D aktivní v H

pwm_LLLL - A a C aktivní v L, B a D aktivní v L

Když používáte výstupy aktivní v H, musíte použít pull-down rezistor mezi piny PICAXE (A-D) a 0V. Když používáte výstupy aktivní v L, musíte použít pull-up rezistor. Účelem rezistoru pull-up nebo pull-down je podržet FET driver ve správném stavu po dobu inicializace PICAXE po zapnutí. V průběhu krátké doby inicializace drivery nejsou aktivně řízeny (tj. „plavou“) a rezistor musí udržet FET v požadovaném stavu.

Režim single (pouze části X1 a X2 – není podporováno na 14M). V režimu single každý pin pracuje nezávisle. Je to tedy ekvivalent příkazu pwmout. Samozřejmě může být v jednom okamžiku zapnutý více než jeden pin.

Tento režim se proto používá zejména ve dvou případech:

1) Jako ekvivalent příkazu **pwmout** na různých výstupech (než příkaz **pwmout**)

2) Pro zapnutí **pwmout** na více než jednom pinu (až na 4) ve stejné době. pwmout aplikovaný na každý výstup je stejný. Toho se často využívá k nastavení jasu na více LED nebo pro řízení více motorů.

Pro zapnutí jediného výstupu jednoduše nastavte jeho odpovídající bit na „1“ (D-C-B-A) v bajtu s nastavením příkazu. Pro zapnutí všech čtyřech pinů například použijte %1111

Režim Half (všechny prvky)

V režimu half výstupy A a C řídí půlmost. C a D se nepoužívají. Signál PWM je na výstupu A, zatímco na výstupu B je komplementární signál PWM. Hodnota nastavení mrtvé zóny je velmi důležitá. Nebude-li správně nastavena, může dojít k současnému sepnutí obou tranzistorů a zničení půlmostu. Toto zpoždění brání v současném zapnutí obou výstupů. Zpoždění (0-127) je násobkem rychlosti oscilátoru, např. hodnota x 4MHz.

4) Tato hodnota závisí na spínací charakteristice použitých tranzistorů FET.

Více informací najdete v datasheetu k použitým driverům.

Režim Full (všechny prvky)

V tomto režimu výstupy A, B, C a D řídí plný most.

V přímém režimu je výstup A nastaven do svého aktivního stavu a výstup D je modulovaný. B a C jsou v neaktivním stavu.

V reverzním režimu je výstup C nastaven do svého aktivního stavu a výstup B je modulovaný. A a D jsou v neaktivním stavu.

V tomto režimu mrtvá zóna (prodleva??????????0) obvykle není vyžadována, protože je v jeden okamžik modulován pouze jeden výstup. Za určitých okolností samozřejmě (někde okolo 100% pracovního cyklu) může nastat současné zapnutí obou výstupů. V takovém případě se doporuje 1) vypnout pwm před zmenou směru nebo 2) použít specializovaný řadič FET, který dokáže zapnout FET rychleji, než se vypne (u nespecializovaných součástek to bývá obráceně).

Více informací najdete v datasheetu k ????????hpwm motor driver.

if…then \ elseif…then \ else \ endif

syntaxe:

IF proměnná ?? hodnota {AND/OR proměnná ?? hodnota …} THEN

{kód}

ELSEIF proměnná ?? hodnota {AND/OR proměnná ?? hodnota …} THEN

{kód}

ELSE

{kód}

ENDIF

- **Proměnné** budou porovnávány s hodnotami.
- **Hodnota** může být zadána jako proměnná nebo konstanta.
- **??** může být jedna z následujících podmínek
  - **=** rovná se
  - **is** rovná se (alternativní zápis)
  - **<>** nerovná se
  - **!=** nerovná se (alternativní zápis)
  - **>** větší než
  - **>=** větší nebo rovno
  - **<** menší než
  - **<=** menší nebo rovno

funkce:

Porovnávání a podmíněné spouštění částí kódu.

Víceřádkový příkaz **if…then\ elseif \ else \ endif** se používá k testování proměnných vstupních pinů (nebo obecných proměnných). Pokud je podmínka splněna, je provedena odpovídající část kódu programu a pak program pokračuje program za pozicí endif. Není-li podmínka splněna, program pokračuje dalším příkazem **elseif** nebo **else**.

Sekce „else“ je provedena pouze tehdy, není-li splněna žádná z podmínek **if** nebo **elseif**.

Jestliže testujete vstupy (pin1, pin2 atd.), musí být uvedeny jejich názvy a ne čísla, tj. např. „if pin1 = 1 then…“, a ne „if 1 = 1 then…“.

Pamatujte si, že

if b0 > 1 then (goto) label ‘(jednořádková struktura)

if b0 > 1 then gosub label ‘(jednořádková struktura)

if b0 > 1 then…else…endif ‘(víceřádková struktura)

jsou 3 zcela oddělené struktury, jež nemohou být kombinovány. Proto je následující řádek neplatný, jelikož se snaží kombinovat jednořádkovou a víceřádkovou strukturu

if b0 > 1 then goto label else goto label2

To nelze, protože kompilátor neví, kterou strukturu se snažíte použít.

Tj.:

if b0 > 1 then goto label : else : goto label2

nebo

if b0 > 1 then : goto label : else : goto label2

Pro použití této struktury musí být řádek přepsán do této podoby

if b0 > 1 then

goto label

else

goto label2

endif

nebo

if b0 > 1 then : goto label : else : goto label2 : endif

Znak : odděluje části, aby kompilátor správně pochopil co má dělat.

if…then {goto}
if…and/or…then {goto}

syntaxe:

IF proměnná ?? hodnota {AND/OR proměnná ?? hodnota …} THEN adresa

- proměnná je proměnná, která bude porovnána s hodnotou

- hodnota může být použita proměnná nebo konstanta

- adresa je návěští, na které se předá řízení programu, je-li podmínka splněna

- ?? může být jeden z následujících operátorů:

=	rovná se
is	rovná se (alternativní zápis)
<>	nerovná se
!=	nerovná se (alternativní zápis)
>	větší než
>=	větší nebo rovno
<	menší než
<=	menší nebo rovno

funkce:

Porovnání a podmíněný skok na jiné místo v programu.

Příkaz if…then se používá ke testování stavu proměnných pro vstupy (nebo obecných proměnných). Jestliže je podmínka splněna, program pokračuje skokem na nové návěští. Není-li podmínka splněna, příkaz je ignorován a program pokračuje na dalším řádku.
Jestliže testujete vstupy (pin1, pin2 atd.), musí být uvedeny jejich názvy a ne čísla, tj. např. „if pin1 = 1 then…“, a ne „if 1 = 1 then…“.
Příkaz if…then kontroluje vstup pouze při zpracování příkazu. Takže se příkaz if…then normálně vkládá do smyčky, která pak pravidelně zjišťuje stav na vstupech. Více informací o tom, jak trvale sledovat vstupy pomocí přerušení najdete v popisu příkazu „setint“.

Například:
Kontrola vstupu uvnitř smyčky.

```
main:      if pin0 = 1 then flsh   ' skok na flsh je-li pin0 nastaven
           goto main               ' jinak skok zpět na start
           flsh: high 1            ' zapnutí výstupu 1
           pause 5000              ' čekání 5 sekund
           low 1                   ' vypnutí výstupu 1
           goto main               ' návrat na start
```

if…then exit
if…and/or…then exit

syntaxe:
IF proměnná ?? hodnota {AND/OR proměnná ?? hodnota …} THEN EXIT
- proměnná je proměnná, která bude porovnána s hodnotou
- hodnota může být použita proměnná nebo konstanta
- adresa je návěští, na něž se předá řízení programu, pokud je podmínka splněna
- ?? může být jeden z následujících operátorů:

=	rovná se
is	rovná se (alternativní zápis)
<>	nerovná se
!=	nerovná se (alternativní zápis)
>	větší než
>=	větší nebo rovno
<	menší než
<=	menší nebo rovno

funkce:
Porovnání a podmíněné ukončení smyčky **do…loop** nebo **for…next**

Příkaz if…then exit se používá k testování proměnných pro vstupy (nebo obecných proměnných). Je-li podmínka splněna, aktuální smyčka je předčasně ukončena (do…loop nebo for…next).
Pomocí klíčových slov AND a OR může být vytvořena složitější podmínka s více porovnáváními. Příklady použití AND a OR najdete v popisu příkazu if…then goto.

Například:
Kontrola vstupu uvnitř smyčky.

```
do
if pin0 = 1 then exit  ' podmíněné ukončení smyčky
loop
```

if…then gosub
if…and/or…then gosub

syntaxe:

IF proměnná ?? hodnota {AND/OR proměnná ?? hodnota …} THEN GOSUB adresa

- **proměnná** je proměnná, jež bude porovnána s hodnotou
- **hodnota** může být použita proměnná nebo konstanta
- **adresa** je návěští, na které se předá řízení programu, pokud je podmínka splněna
- **??** může být jeden z následujících operátorů:
  - **=** rovná se
  - **is** rovná se (alternativní zápis)
  - **<>** nerovná se
  - **!=** nerovná se (alternativní zápis)
  - **>** větší než
  - **>=** větší nebo rovno
  - **<** menší než
  - **<=** menší nebo rovno

funkce:

Porovnání a podmíněné provedení příkazu gosub.

Příkaz if…then gosub se používá k testování proměnných pro vstupy (nebo obecných proměnných). Je-li podmínka splněna, je proveden podprogram. Když podmínka splněna není, příkaz je ignorován a program pokračuje na dalším řádku. Po provedení podprogramu program pokračuje na dalším řádku.

Při použití vstupů musí být použita proměnná pro vstupy (pin1, pin2 atd.) a ne název pinu (1, 2 atd.), takže řádek musí vypadat takto: „if pin1 = 1 then gosub…“ a ne „if 1 = 1 then gosub…“

Příkaz if…then gousb kontroluje vstup pouze v době svého provedení. Proto se vkládá do smyčky programu, která pak pravidelně čte vstupy.

Podmínky je možno kombinovat pomocí klíčových slov AND a OR. Příklady použití AND a OR najdete u příkazu if…then goto.

Například:

Kontrola vstupu ve smyčce.

```
main:      if pin0 = 1 then gosub flsh   ' podpogram flsh je-li pin0 v logické 1
goto main                              ' jinak zpátky na start
flsh: high 1                           ' zapnutí výstupu 1
pause 5000                             ' čekání 5 sekund
low 1                                       ' vypnutí výstupu 1
return
```

2vstupové hradlo AND

```
if pin1 = 1 and pin2 = 1 then gosub label
```

3vstupové hradlo AND

```
if pin0 =1 and pin1 =1 and pin2 = 1 then gosub label
```

2vstupové hradlo OR

```
if pin1 =1 or pin2 =1 then gosub label
```

analogicky hodnota mezi určitými hodnotami

```
readadc 1,b1
if b1 >= 100 and b1 <= 200 then gosub label
```

Pro čtení celého portu najednou je možno použít proměnnou „pins“

```
if pins = %10101010 then gosub label
```

Čtení celého portu a maskování jendotlivých vstupů (např. 6 a 7)

```
let b1 = pins & %11000000
if b1 = %11000000 then gosub label
```

Mladší studenti mohou používat také slova is (=), on (1) a off (0).

```
loop1:     if pin0 is on then gosub flsh ' flsh je-li pin0 v log. 1
           goto loop1                    ' jinak zpátky na start
           flsh: high 1                  ' zapnout výstup 1
           pause 5000                    ' čekat 5 sekund
           low 1                         ' vypnout výstup 1
           return                        ' návrat
```

if … then
if … and … then
if … or … then

syntaxe:

IF variable ?? value {AND/OR variable ?? value …} THEN address

- **variable** je proměnná. Bude porovnána s value
- **value** může být proměnná nebo konstanta
- **adresa** je návěští, na které se předá řízení programu, pokud je podmínka splněna
- **??** může být jeden z následujících operátorů:

=	rovná se
is	rovná se (alternativní zápis)
<>	nerovná se
!=	nerovná se (alternativní zápis)
>	větší než
>=	větší nebo rovno
<	menší než
<=	menší nebo rovno

funkce:

Příkaz porovnává dvě proměnné nebo proměnnou s konstantou a skočí na určené místo, pokud je podmínka splněna. Pokud není podmínka splněna, pokračuje se na daším řádku programu.

Například:

```
main:      if pin0 = 1 then flsh     ' jump to flsh if pin0 is high
           goto main                 ' else loop back to start
flsh:      high 1                    ' switch on output 1
           pause 5000                ' wait 5 seconds
           low 1                     ' switch off output 1
           goto main                 ' loop back to start
```

Vícenásobná podmínka může být vytvořena pomocí AND a OR:

2 násobné AND – musí být splněny všechny podmínky současně, aby se skočilo na label

```
if pin1 = 1 and pin2 = 1 then label
```

3 násobné AND – musí být splněny všechny podmínky současně, aby se skočilo na label

```
if pin0 = 1 and pin1 = 1 and b0 > 3 then label
```

2 násobné OR – musí být splněna alespoň jedna podmínka (nebo obě dvě), aby se skočilo na label

```
if pin1 = 1 or b3 = b2 then label
```

testování analogové hodnoty v daném intervalu, b1 musí být mezi 100 a 200, aby se skočilo na label

```
readadc 1,b1
if b1 >= 100 and b1 <= 200 then label
```

inc

syntaxe:
INC proměnná
- vstupem je proměnná, jejíž obsah se má inkrementovat, čili zvýšit o 1

funkce:
Inkrementování (přičtení 1 k) obsahu proměnné.

Tento příkaz je zkráceným zápisem pro „let var = var + 1“

Například:
```
for b1 = 1 to 5
inc b2
next b1
```

infrain2

syntaxe:
INFRAIN2

funkce:
Obdoba předchozího příkazu pro PICAXE-08M. Vyžaduje připojení přijímače infračerveného signálu na vstup input3, používá protokol Sony.
Podrobnosti naleznete v originální anglické příručce.

infraout

syntaxe:
INFRAOUT device,data

funkce:
Vyšle data protokolem Sony infračerveného dálkového ovládání, pouze na PICAXE-08M. Předpokládá připojení vysílací infračervené LED s omezovacím odporem na výstup 0.
Podrobnosti naleznete v originální anglické příručce.

input (použitelné pro PICAXE 08M)

syntaxe:
INPUT pin
- pin je proměnná nebo konstanta, označuje použitý vývod

funkce:
Nastaví vývod jako vstupní.
Po zapnutí napájení jsou všechny konfigurovatelné vývody nastaveny jako vstupní. Kromě příkazů k přímému nastavení (LET DIRS, INPUT, OUTPUT, REVERSE) se příslušné vývody nastavují také příkazy HIGH, LOW, TOGGLE, PULSOUT jako výstupní.
U kontrolérů 08 a 08M je pin0 vždy výstupní a pin3 vždy vstupní.

Například:
```
main:     input 1          ' make pin input
          reverse 1        ' make pin output
          reverse 1        ' make pin input
          output 1         ' make pin output
```

let

syntaxe:

{LET} variable = {-} value ?? value…

- variable je proměnná, které bude přiřazen výsledek početní operace

- value jsou proměnné nebo konstanty spojené operátory

Klíčové slovo LET je nepovinné

funkce:

Provádí početní operace v celočíselné 16-bitové aritmetice (hodnoty 0 až 65535). Výrazy jsou vyhodnocovány zleva doprava bez upatnění přednosti operátorů. Všechna čísla jsou chápána jako kladná. Výsledek může být také 8-bitový nebo 1-bitový, v tom případě jsou vyšší bity oříznuty.

Například:

```
main:      let b0 = b0 + 1          ' increment b0
           sound 7,(b0,50)          ' make a sound
           if b0 > 50 then rest     ' after 50 reset
           goto main                ' loop back to start
rest:      let b0 = b0 max 10       ' limit b0 back to 10
                                    ' as 10 is the maximum value
           goto main                ' loop back to start
```

let dirs =
let dirsc =

syntaxe:

{LET} dirs = value

{LET} dirsc = value

- value jsou proměnné nebo konstanty, jejichž hodnota je uložena do proměnné dirs (dirsc).

funkce:

Konfiguruje vývody kontroléru jako vstupní nebo výstupní (let dirs, u PICAXE-08M)

Konfiguruje vývody kontroléru na portu C jako vstupní nebo výstupní (let dirsc, u PICAXE-14M). Hodnota 1 příslušného bitu značí vstup, hodnota 0 značí výstup.

Po zapnutí napájení jsou všechny konfigurovatelné vývody nastaveny jako vstupní. Kromě příkazů k přímému nastavení (LET DIRS, INPUT, OUTPUT, REVERSE) se příslušné vývody nastavují také příkazy HIGH, LOW, TOGGLE, PULSOUT jako výstupní.

U kontrolérů 08M a 14M je pin0 vždy výstupní a pin3 vždy vstupní.

Například:

```
let dirs = %00000011          ' switch pins 0 and 1 to outputs
let pins = %00000011          ' switch on outputs 0 and 1
```

let pins =
let pinsc =

syntaxe:

{LET} pins = value

{LET} pinsc = value

- value je proměnná nebo konstanta, jejíž hodnota je uložena do proměnné pins (pinsc).

funkce:

Tento příkaz nastavuje současně všechny výstupy kontroléru individuálně na vysokou nebo nízkou úroveň, u kontrolérů PICAXE-14M lze takto nastavit též všechny vývody portu C. Klíčové slovo LET je nepovinné.

K individuálnímu nastavení jednotlivých výstupů lze použít příkazy high a low. Příkaz LET PINS umožňuje hromadné nastavení všech osmi výstupů současně. S výhodou se konstanty uvádí v binárním tvaru, potom výstupu 7 odpovídá číslice zcela vlevo a výstupu 0 číslice na posledním místě. Uvedením hodnoty 0 se příslušný výstup nastaví na nízkou úroveň, hodnotou 1 se nastaví na vysokou úroveň.

Proměnná pins, pokud je čtena (vpravo od rovnítka), odpovídá vstupům. Při zápisu do proměnné pins odpovídá výstupům. Z tohoto důvodu není možné například uvést výstupy 0 až 3 do nízké úrovně příkazem LET PINS=PINS & %11110000.

Na kontrolérech s obousměrnými vývody (08M a 14M) se tento příkaz aplikuje pouze na vývody, které jsou nastaveny jako výstupní. Příslušné vývody lze aktivovat jako výstupy příkazem LET DIRS.

Například:

```
let pins = %11000011          ' switch outputs 7,6,0,1 on
pause 1000                    ' wait 1 second
let pins = %00000000          ' switch all outputs off
```

lookdown

syntaxe:

LOOKDOWN target,(value0,value1…valueN),variable

- target je proměnná nebo konstanta, která se porovnává s řadou hodnot v závorce.

- value0… jsou proměnné nebo konstanty

- variable obsahuje výsledek porovnávání.

funkce:

Porovnává target se seznamem hodnot v závorce, pokud najde stejnou hodnotu, uloží do proměnné za závorkou pořadové číslo shodné hodnoty. Číslování začíná nulou. Pokud se shoda nenajde, proměnná za závorkou zůstane beze změny.

Například:

```
lookdown b1,("abcde"),b2
```

lookup

syntaxe:

LOOKUP offset,(data0,data1…dataN),variable

- offset je proměnná nebo konstanta, určuje, která položka z data0 až dataN se uloží do proměnné variable.

- data jsou proměnné nebo konstanty

- variable předává výslednou hodnotu, nebo zůstává beze změny

funkce:

Vybírá z pole hodnot podle zadaného ofsetu (indexu). Pokud je ofset mimo rozsah uvedených hodnot, výstupní proměnná zůstává beze změny.

Například:

```
main:       let b0 = b0 + 1           ' increment b0
            lookup b0,("abcd"),b1     ' put ascii character into b1
            if b0 < 4 then loop       ' loop
            end
```

low

syntaxe:

LOW pin

- pin je proměnná nebo konstanta, označuje výstup, který se použije

funkce:

Nastaví nízkou výstupní úroveň. (U PICAXE-08M a 14M zároveň nastaví vývod jako výstupní.)

Například:

```
main:     high 1            ' switch on output 1
          pause 5000        ' wait 5 seconds
          low 1             ' switch off output 1
          pause 5000        ' wait 5 seconds
          goto main         ' loop back to start
```

low portc (použitelné pro PICAXE 14M)

syntaxe:

LOW PORTC pin

- pin je proměnná nebo konstanta, označuje výstup na portu C, který se použije

funkce:

Nastaví nízkou výstupní úroveň na výstupu portu C.

Například:

```
main:     high portc 1      ' switch on output 1
          pause 5000        ' wait 5 seconds
          low portc 1       ' switch off output 1
          pause 5000        ' wait 5 seconds
          goto main         ' loop back to start
```

nap

syntaxe:

NAP period

- period je proměnná nebo konstanta, určující dobu, na kterou přejde kontroler do režimu s nízkou spotřebou. Rozsah 0 až 7.

Doba zpoždění:

0	18 ms
1	32 ms
2	72 ms
3	144 ms
4	288 ms
5	576 ms
6	1,152 s
7	2,304 s

funkce:

Uvede kontroler do spánku na dobu 2^period*18 ms. Tento příkaz využívá watchdog timer s omezenou přesností časování. Delší prodlevy lze dosáhnout příkazem Sleep.

Například:

```
main:     high 1       ' switch on output 1
          nap 4        ' nap for 288ms
          low 1        ' switch off output 1
          nap 7        ' nap for 2,3 s
          goto main    ' loop back to start
```

on…goto

syntaxe:

ON ofset GOTO adresa0,adresa1…adresaN

- ofset je proměnná nebo konstanta udávající adresu, jež má být použita (0-N).

- adresy návěští, na která se má přejít.

funkce:

Větvení programu na adresu danou ofsetem (pakliže je v rozsahu).

Tento příkaz umožňuje skok na různá místa v programu v závislosti na hodnotě proměnné „ofset“. Jestliže je ofset 0, program bude pokračovat na návěští adresa0, bude=li ofset 1, program bude pokračovat na návěští adresa1 atd.

Když bude ofset vyšší než počet adres uvedených v příkazu, bude příkaz ignorován a program bude pokračovat na dalším řádku.

Tento příkaz je funkčně identický s větvením

Například:

```
reset:      let b1 = 0
            low 0
            low 1
            low 2
            low 3
main:       let b1 = b1 + 1
            if b1 > 3 then reset
            on b1 goto btn0,btn1, btn2, btn3
            goto main
btn0:       high 0
            goto main
btn1:       high 1
            goto main
btn2:       high 2
            goto main
btn3:       high 3
            goto main
```

on…gosub

syntaxe:

ON ofset GOSUB adresa0,adresa1…adresaN

- **ofset** je proměnná nebo konstanta udávající, který podprogram se má použít (0-N).
- **adresy** návěští podprogramů.

funkce:

Skok do podprogramu daného ofsetem (je-li v rozsahu).

Tento příkaz umožňuje podmíněný skok do podprogramu v závislosti na hodnotě proměnné „ofset“. Je-li ofset 0, program zavolá podprogram na návěští adresa0, je-li ofset 1, program zavolá podprogram na návěští adddress1 atd.

Bude-li ofset vetší než počet adres, bude celý příkaz ignorován a program bude pokračovat na dalším řádku.

Po provedení příkazu return v podprogramu bude program pokračovat na dalším řádku za příkazem on…gosub

Například:

```
reset:      let b1 = 0
            low 0
            low 1
            low 2
            low 3
main:       let b1 = b1 + 1
            if b1 > 3 then reset
            on b1 gosub btn0,btn1, btn2, btn3
            goto main
btn0:       high 0
            return
btn1:       high 1
            return
btn2:       high 2
            return
btn3:       high 3
            return
```

output (použitelné pro PICAXE 08M)

syntaxe:

OUTPUT pin

- pin je proměnná nebo konstanta, označuje použitý vývod

funkce:

Nastaví vývod jako výstupní. Funguje pouze u kontrolérů 08 a 08M.

Například:

```
main:       input 1      ' make pin input
            reverse 1    ' make pin output
            reverse 1    ' make pin input
            output 1     ' make pin output
```

Po zapnutí napájení jsou všechny konfigurovatelné vývody nastaveny jako vstupní. Kromě příkazů k přímému nastavení (LET DIRS, INPUT, OUTPUT, REVERSE) se příslušné vývody nastavují také příkazy HIGH, LOW, TOGGLE, PULSOUT jako výstupní.

U kontrolérů 08, 08M a 14M je pin0 vždy výstupní a pin3 vždy vstupní.

pause

syntaxe:

PAUSE milliseconds

- milliseconds je proměnná nebo konstanta v rozsahu 0 až 65535, určuje dobu v jednotkách milisekund, po kterou bude tento příkaz trvat. Toto platí pouze při nastavené hodinové frekvenci 4 MHz. Při nastavení hodinové frekvence 8 MHz se čas zkracuje na 0,5 ms a na 0,25 ms při nastavení hodinové frekvence na 16 MHz.

funkce:

Zastaví běh programu na určenou dobu. Přesnost je odvozena od hodinového kmitočtu kontroleru.

Například:

```
main:       high 1       ' switch on output 1
            pause 5000   ' wait 5 seconds
            low 1        ' switch off output 1
            pause 5000   ' wait 5 seconds
            goto main    ' loop back to start
```

peek

syntaxe:

PEEK location,variable

- location je proměnná nebo konstanta, určující adresu registru. Platné hodnoty jsou 0 až 255.

- variable je 8bitová proměnná, ve které je navrácen obsah registru na udané adrese.

funkce:

Čte data z registrů mikrokontroleru. Umožňuje obnovit data uschovaná příkazem POKE.

Například:

```
peek 80,b1       ' put value of register 80 into variable b1
```

play

syntaxe:

PLAY tune,LED

- tune je proměnná nebo konstanta (0 - 3) určující, která skladba se zahraje:
0 - Happy Birthday
1 - Jingle Bells
2 - Silent Night
3 - Rudolf the Red Nosed Reindeer

- LED je proměnná nebo konstanta (0 -3) určující způsob blikání připojených LED během hraní:
0 - bez blikání
1 - výstup 0 se zapíná a vypíná
2 - výstup 4 se zapíná a vypíná
3 - výstupy 0 a 4 se střídavě zapínají a vypínají

funkce:

Přehrává skladbu. Výstupní signál se objeví na výstupu 2.

Například:

```
play 3,1   ' rudolf red nosed reindeer with output 0 flashing
```

poke

syntaxe:

POKE location,data

- location je proměnná nebo konstanta určující adresu registru. Platné hodnoty jsou 0 až 255.

- data je proměnná nebo konstanta obsahující data, která budou zapsána na uvednou adresu.

funkce:

Zapisuje data do registrů kontroleru. Umožňuje uložit proměnné b0 až b13 do paměti a dále využít hardware prostřednictvím SFR (podle dokumentace v katalogovém listu příslušného kontroleru).

Například:

```
poke 80,b1       ' save value of b1 in register 80
```

pulsin

syntaxe:

PULSIN pin,state,variable

- pin je proměnná nebo konstanta (0-7), určující, který vývod bude použit.

- state je proměnná nebo konstanta (0 nebo 1), určující, která hrana se musí první objevit před začátkem měření.

- variable obsahuje výsledek měření (1 - 65535) v jednotkách 10 µs.

funkce:

Měří délku vstupního pulsu v jednotkách 10 µs. Jestliže se puls neobjeví do 0,65536 s, příkaz končí a výsledek je 0. Pokud je proměnná state = 1, měří se délka pulsu ve vysoké úrovni, měření začíná vzestupná hrana a končí sestupná hrana. Pokud je proměnná state = 0, měří se délka pulsu v nízké úrovni, měření začíná sestupná hrana a končí vzestupná hrana. Výstupní proměnná se obvykle používá 16bitová.

Vliv frekvence oscilátoru:

4 MHz – doba v jednotkách 10µs a ukončení příkazu po 0,65536 s
8 MHz – doba v jednotkách 5µs a ukončení příkazu po 0,32768 s
16 MHz – doba v jednotkách 5µs a ukončení příkazu po 0,16384 s

Například:

```
pulsin 3,1,w1    ' record the length of a pulse on pin 3 into b1
```

pulsout

syntaxe:

PULSOUT pin,time

- pin je proměnná nebo konstanta (0-7) určující, který vývod bude použit.
- time je proměnná nebo konstanta určující dobu trvání pulsu (0-65535) v jednotkách 10 µs.

funkce:

Vyšle impuls zadané délky. Polarita impulsu je určená počátečním stavem pinu, během pulsu se stav invertuje a po skončení se vrátí na původní úroveň

Vliv frekvence oscilátoru:

4 MHz doba (time) v jednotkách 10 µs
8 MHz doba (time) v jednotkách 5 µs
16 MHz doba (time) v jednotkách 2,5 µs

Například:

```
main:      pulsout 4,150     ' send a 1,50 ms pulse out of pin 4
           pause 20          ' pause 20 ms
           goto main         ' loop back to start
```

pwm (použitelné pro PICAXE 08M)

syntaxe:

PWM pin,duty,cycles

- pin je proměnná nebo konstanta (0-7), určující, který vývod bude použit.
- duty je proměnná nebo konstanta (0-255), určující činitel plnění PWM (dobu, kterou výstup setrvá ve stavu 1)
- cycles je proměnná nebo konstanta (0-255), určující počet cyklů PWM, které na určeném vývodu proběhnout. Každý cyklus trvá asi 5 ms.

funkce:

Tento příkaz se používá zřídka, vhodnější je použít PWMOUT. Příkaz PWM ukončí svoji činnost po zadaném počtu cyklů, neprobíhá na pozadí jako příkaz PWMOUT. Ve spojení s RC filtrem může napodobit analogový výstup na kontroléru PICAXE-08. Příkaz musí být volán opakovaně.

Například:

```
main:      pwm 4,150,20      ' send 20 pwm bursts out of pin 4
           pause 20          ' pause 20 ms
           goto main         ' loop back to start
```

pwmout

syntaxe:

PWMOUT pin,period,duty cycles

- pin je proměnná nebo konstanta určující, který vývod bude použit.
(pouze vývod 2 na PICAXE 08M a 14M)

- period je proměnná nebo konstanta (0-255) určující periodu pulsně šířkové modulace (PWM)

- duty je proměnná nebo konstanta (0-1023) určující činitel plnění PWM (dobu, kterou výstup setrvá ve stavu 1)

Generuje pulsně modulovaný výstupní signál na zvoleném vývodu s využitím interního hardware kontroleru. Tento signál zůstává aktivní i po ukončení příkazu PWMOUT. Pokud je třeba signál zrušit, použije se příkaz PWMOUT s periodou 0.

Perioda PWM	= (period + 1) * 4 * perioda oscilátoru
Činitel plnění PWM	= (duty) * perioda oscilátoru
	Perioda oscilátoru je 250 ns při 4 MHz, 125 ns při 8 MHz a 62,5 ns při 16 MHz
Frekvence PWM	= 1 / perioda PWM

Tento příkaz používá hardware kontroleru, z čehož plynou následující omezení.

1) Příkaz funguje pouze na vývodu 2 u 08M a 14M.

2) Činitel plnění je 10 bitová hodnota, která by neměla být větší nežli čtyřnásobek periody PWM (je-li větší, PWM výstup zůstává stále na vysoké úrovni).

3) Příkaz Servo nemůže být aktivní zároveň s příkazem PWMout, neboť využívá tentýž časovač.

4) PWM výstup se deaktivuje během příkazů nap, sleep, a po provedení příkazu end.

Například:

```
main:     pwmout 2,150,100  ' set pwm
          pause 1000        ' pause 1 s
          goto main         ' loop back to start
```

random

syntaxe:

RANDOM wordvariable

- wordvariable slouží zároveň jako výsledek i jako pracovní proměnná (násada) pro příští použití příkazu. Musí být použita proměnná typu word a její hodnota se nesmí do dalšího použití příkazu změnit.
Vytváří sekvenci pseudonáhodných čísel mezi 0 a 65535.

Například:

```
main:     random w0               ' put random value into w0
          if pin1 =1 then doit
          goto main
doit:     let pins = b1           ' put random byte value on output pins
          pause 100               ' wait 0.1s
          goto main               ' loop back to start
```

readadc

syntaxe:

READADC channel,variable

- channel je proměnná nebo konstanta, určující vstup (0-7)

- variable obsahuje výsledek A/D převodu

funkce:

Příkaz čte napětí na analogovém vstupu a převádí ho na osmibitové číslo. Pouze některé vstupy mohou sloužit jako analogové. Na některých kontrolerech jsou analogové a digitální vstupy sdílené, mohou plnit obě funkce.

Například:

```
main:     readadc 1,b1            ' read value into b1
          if b1 > 50 then flsh    ' jump to flsh if b1 > 50
          goto main               ' else loop back to start
flsh:     high 1                  ' switch on output 1
          pause 5000              ' wait 5 seconds
          low 1                   ' switch off output 1
          goto main               ' loop back to start
```

readadc10

syntaxe:

READADC10 channel,wordvariable

- channel je proměnná nebo konstanta určující vstup (0-7)

- wordvariable obsahuje výsledek A/D převodu

funkce:

Příkaz čte napětí na analogovém vstupu a převádí ho na 10bitové číslo, proto se výsledek musí ukládat do 16bitové proměnné. Pouze některé vstupy mohou sloužit jako analogové. Na některých kontrolerech jsou analogové a digitální vstupy sdílené, mohou plnit obě funkce. Při použití příkazu DEBUG může komunikace s PC narušit výsledek A/D převodu. V tom případě se doporučuje doplnit programovací obvod o Schottkyho diodu, která tento efekt potlačí.

Například:

```
main:     readadc 1,w1            ' read value into b1
          debug w1                ' transmit to computer
          pause 200               ' short delay
          goto main               ' loop back to start
```

read

syntaxe:

READ location,variable

- location je proměnná nebo konstanta, určující 8bitovou adresu v interní EEPROM (0-255).

- variable obsahuje přečtená data

funkce:

Příkaz READ načítá data z EEPROM. Obsah této paměti je zachován i po vypnutí napájení. Tato data jsou zapisována při každém zavedení nového programu do mikrokontroléru, podle definice v příkazu DATA/EEPROM. Za běhu programu mohou být tato data přepisována pomocí příkazu WRITE. Příkaz READ pracuje pouze s osmibitovými daty. Pokud je třeba načíst proměnnou word, dosáhne se toho použitím dvou příkazů READ s oběma osmibitovými proměnnými: (například w0 se načte tak, že se načte b0 a b1). Velikost, umístění a případná omezení použitelnosti eeprom u jednotlivých typů kontrolérů jsou uvedeny u příkazu DATA.

Například:

```
main:      for b0 = 0 to 63          ' start a loop
           read b0,b1                ' read value into b1
           serout 7,T2400,(b1)       ' transmit value to serial LCD
           next b0                   ' next loop
```

readoutputs

syntaxe:

READOUTPUTS proměnná

- proměnná je proměnná typu byte pro uložení hodnot výstupních pinů

funkce:

Načtení stavu výstupních pinů do proměnné.

Aktuální stav výstupních pinů může být načten do proměnné pomocí příkazu readoutputs. Pamatujte si, že to není totéž jako „let var = pins“, jelikož tento příkaz let načítá stav vstupních a ne výstupních pinů. Tento příkaz se normálně nepoužívá s X1 a X2, jelikož výstupy mohou být načteny přímo pomocí „let var = outpins“

Například:

```
main:      readoutputs b1    ' načtení hodnot výstupů do proměnné b1
```

readtemp

syntaxe:

READTEMP pin,variable

- pin je proměnná nebo konstanta, určující, který vývod bude použit.

- variable obsahuje přečtená data (byte).

funkce:

Příkaz přečte teplotu z digitálního čidla DS18B20 a uloží ji do proměnné. Převod může trvat až 750 ms. Teplota se předává v celých stupních celsia. Senzor pracuje v rozmezí teplot -55 až +125°C. Bit 7 představuje znaménko, je 0 pro teploty nad nulou a 1 pro teploty pod nulou, záporné teploty se vrací jako 128 + teplota bez znaménka.

Příkaz readtemp nefunguje se staršími čidly DS1820 nebo DS18S20, která mají odlišný formát dat. Tento příkaz funguje pouze při frekvenci 4 MHz.

Například:

```
main:       readtemp 1,b1           ' read value into b1
            if b1 > 127 then neg    ' test for negative
            serout 7,N2400,(#b1)    ' transmit value to serial LCD
            goto main
neg:        let b1 = b1 - 128       ' adjust neg value
            serout 7,N2400,("-")    ' transmit negative symbol
            serout 7,N2400,(#b1)    ' transmit value to serial LCD
            goto main
```

readtemp12

syntaxe:

READTEMP12 pin,wordvariable

- pin je proměnná nebo konstanta, určující, který vývod bude použit.

- wordvariable obsahuje přečtená data (12 bitů).

funkce:

Příkaz přečte teplotu v surovém 12 bitovém formátu z digitálního čidla DS18B20 a uloží ji do proměnné. Převod může trvat až 750 ms. Podrobnosti o formátu dat lze nalézt v dokumentaci k teplotnímu čidlu.

Příkaz readtemp12 nefunguje se staršími čidly DS1820 nebo DS18S20, která mají odlišný formát dat.

Tento příkaz funguje pouze při frekvenci 4 MHz.

Například:

```
main:       readtemp12 1,w1         ' read value into b1
            debug w1                ' transmit to computer screen
            goto main
```

readowsn (použitelné pro PICAXE 08M)

READOWSN pin

syntaxe:

- pin je proměnná nebo konstanta (0-7), určující, který vývod bude použit.

funkce:

Přečte sériové číslo z obvodu připojeného na jednodrátovou sběrnici firmy Dallas. Může číst například z teplotního senzoru DS18B20, obvodu reálného času DS2415 nebo identifikačního obvodu DS1990A (iButton). U DS1990A je sériové číslo také vypálené laserem na pouzdru obvodu.

Příkaz ukládá přečtená data následovně - family code do proměnné b6, sériové číslo do proměnných b7 až b12 a kontrolní součet do b13.

Například:

```
main:       resetowclk 2        ' reset the clock on pin2
main:       readowclk 2         ' read clock on input2
            debug b1            ' display the elapsed time
            pause 10000         ' wait 10 seconds
            goto main
```

return

syntaxe:

RETURN

funkce:

Návrat z podprogramu. Příkaz return smí být použit pouze po předchozím příkazu gosub. Příkaz navrací běh programu do místa, odkud byl podprogram vyvolán. Pokud by byl příkaz return použit bez předchozího gosub, program havaruje.

Například:

```
main:     let b2 = 15       ' set b2 value
          pause 2000        ' wait for 2 seconds
          gosub flsh        ' call sub-procedure
          let b2 = 5        ' set b2 value
          pause 2000        ' wait for 2 seconds
          gosub flsh        ' call sub-procedure
          end               ' stop accidentally falling into sub
flsh:     for b0 = 1 to b2  ' define loop for b2 times
          high 1            ' switch on output 1
          pause 500         ' wait 0.5 seconds
          low 1             ' switch off output 1
          pause 500         ' wait 0.5 seconds
          next b0           ' end of loop
          return            ' return from sub-procedure
```

reverse (použitelné pro PICAXE 08M)

syntaxe:

REVERSE pin

- pin je proměnná nebo konstanta, určující, který vývod bude použit.

funkce:

Změní směr signálu na vývodu – původně vstup nastaví jako výstup a původně výstup nastaví jako vstup. Funguje pouze u kontrolérů 08 a 08M.

Po zapnutí napájení jsou všechny konfigurovatelné vývody nastaveny jako vstupní. Kromě příkazů k přímému nastavení (LET DIRS, INPUT, OUTPUT, REVERSE) se příslušné vývody nastavují také příkazy HIGH, LOW, TOGGLE, PULSOUT jako výstupní.

U kontrolérů 08 a 08M je pin0 vždy výstupní a pin3 vždy vstupní.

Například:

```
main:     input 1      ' make pin input
          reverse 1    ' make pin output
          reverse 1    ' make pin input
          output 1     ' make pin output
```

select case \ case \ else \ endselect

syntaxe:

SELECT proměnná

CASE hodnota

{kód}

CASE hodnota, hodnota…

{kód}

CASE hodnota TO hodnota

{kód}
CASE ?? hodnota
{kód}
ELSE
{kód}
ENDSELECT

- **proměnná** je proměnná. Bude porovnána s hodnotami
- **hodnota** může být proměnná nebo konstanta
- **adresa** je návěští, na které se předá řízení programu, je-li podmínka splněna
- **??** může být jeden z následujících operátorů:
  - **=** rovná se
  - **is** rovná se (alternativní zápis)
  - **<>** nerovná se
  - **!=** nerovná se (alternativní zápis)
  - **>** větší než
  - **>=** větší nebo rovno
  - **<** menší než
  - **<=** menší nebo rovno

funkce:
Porovnání hodnoty proměnné a podmíněné provedení sekcí programu.

Pro testování různých podmínek se používají různé formy zápisu select \ case \ else \ endselect. Jsou-li tyto podmínky splněny, jsou provedeny odpovídající části kódu programu a program pak pokračuje za endselect. Není-li podmínka splněna, program pokračuje další klauzulí case nebo příkazem else.
Sekce „else“ je provedena pouze pokud nebyla splněna žádná z podmínek.

Například:

```
select case b1
case 1
high 1
case 2,3
low 1
case 4 to 6
high 2
else
low 2
endselect
```

serin

syntaxe:
SERIN pin,baudmode,(qualifier,qualifier…)
SERIN pin,baudmode,(qualifier,qualifier…),{#}variable,{#}variable…
SERIN pin,baudmode,{#}variable,{#}variable…

- **pin** je proměnná nebo konstanta (0-7) určující, který vývod bude použit.
- **baudmode** je proměnná nebo konstanta (0-7) která určuje přenosovou rychlost a polaritu signálu.

Všechny přenosové rychlosti se vztahují k hodinovému kmitočtu 4 MHz:

T2400 normální polarita (True, klidová úroveň vysoká)
T1200 normální polarita
T600 normální polarita
T300/T4800 normální polarita
N2400 obrácená polarita (Negated, klidová úroveň nízká)
N1200 obrácená polarita
N600 obrácená polarita
N300/N4800 obrácená polarita

- qualifier je nepovinná proměnná nebo konstanta, která musí být přijata v určeném pořadí nežli se začnou zpracovávat data a ukládat do proměnné (proměnných).

- variable obdrží přijatý znak (0-255). Nepovinné znak # signalizuje, že se mají následující znaky interpretovat jako číslo v desítkové soustavě a uložit do proměnné tuto hodnotu, nikoliv kód znaku.

funkce:

Tento příkaz přijímá sériová data pouze ve formátu 8N1 (8 datových bitů, bez parity, 1 stop bit).

Příkaz serin příjímá sériová data na určeném vstupu. Nelze ho použít k příjmu na vstupu Sin, určeném pro zavedení programu.

Pin určuje na kterém vstupu se budou sériová data přijímat.

Baudmode určuje přenosovou rychlost a polaritu signálu. Pokud je signál úrovně RS232 přiveden pouze přes omezovací odpor, používá se polarita N (obrácená). Pokud je signál zpracováván obvodem typu MAX232, používá se polarita T (normální).

Přenosová rychlost 4800 je dostupná pouze na kontrolérech řady -X. Při této přenosové rychlosti se může stát, že kontrolér nebude schopen přijímat komplikované protokoly – zpracování dat bude příliš pomalé. Doporučuje se maximální přenosová rychlost 2400 při hodinovém kmitočtu 4MHz.

Qualifier značí specifický znak nebo sequenci znaků (řetězec), který musí být přijat nežli se začnou zpracovávat další znaky a plnit proměnné. Například příkaz:

```
serin 1,N2400,("ABC"),b1
```

čeká nejprve na řetězec “ABC” a když je přijat, uloží další znak do proměnné b1.

Příkaz bez uvedení qualifier

```
serin 1,N2400,b1
```

uloží do proměnné b1 první přijatý znak.

Během příkazu serin jsou veškeré další aktivity kontroleru zastaveny, dokud není přijat nějaký znak. Tento příkaz není také přerušen pomocí setint.

Následující příklad čeká, dokud není přijat řetězec “go”.

```
serin 1,N2400,("go")
```

Při použití příkazu serin může být nutné resetovat kontrolér pomocí vstupu reset (MCLR), aby započalo zavedení nového programu.

Například:

```
main:       for b0 = 0 to 63          ' start a loop
            serin 6,N2400,b1          ' receive serial value
            write b0,b1               ' write value into b1
            next b0                   ' next loop
```

serout

syntaxe:

SEROUT pin,baudmode,({#}data,{#}data…)

- pin je proměnná nebo konstanta (0-7) určující, který vývod bude použit.

- baudmode je proměnná nebo konstanta (0-7) která určuje přenosovou rychlost a polaritu signálu.

T2400 normální polarita ([T]rue, klidová úroveň vysoká)
T1200 normální polarita
T600 normální polarita
T300 / T4800 normální polarita
N2400 obrácená polarita ([N]egated, klidová úroveň nízká)
N1200 obrácená polarita
N600 obrácená polarita
N300 / N4800 obrácená polarita

- data jsou proměnné nebo konstanty (0-255), jejichž hodnoty budou vyslány na určený výstup.

Nepovinný znak # signalizuje, že se následující hodnota má odeslat jako číslo v desítkové soustavě, nikoliv jako jeden znak. Textový řetězec může být uveden v uvozovkách („Hello“).

Vysílá sériová data ve formátu 8N1 (8 datových bitů, bez parity, 1 stop bit).

funkce:

Příkaz serout vysílá data asynchronním sériovým přenosem na určeném výstupu mikrokontroléru. Nelze ho použít s vývodem Sout, určeném ke komunikaci při zavádění nového programu. Na tomto vývodu pracuje příkaz sertxd.

Pin určuje na kterém vstupu se budou sériová data vysílat.

Baudmode určuje přenosovou rychlost a polaritu signálu. Pokud je signál úrovně RS232 přiveden pouze přes omezovací odpor, používá se polarita N (obrácená). Pokud je signál zpracováván obvodem typu MAX232, používá se polarita T (normální).

Symbol # značí konverzi na dekadický retězec. Například když b1 obsahuje hodnotu 126, potom #b1 způsobí vyslání znaků „1“ „2“ a „6“ namísto znaku s kódem 126.

Například:

```
main:     for b0 = 0 to 63          ' start a loop
          read b0,b1                ' read value into b1
          serout 7,N2400,(b1)       ' transmit value to serial LCD
          next b0                   ' next loop
```

sertxd

syntaxe:

SERTXD ({#}data,{#}data…)

- data jsou proměnné nebo konstanty (0-255), jejichž hodnoty budou vyslány na výstup Sout.

funkce:

Asynchronní sériový výstup na vývodu Sout určeném ke komunikaci při zavádění nového programu. Přenosová rychlost je 4800Bd, 8N1. Tato data lze sledovat na PC v okně terminálu, které se otevře příkazem Terminal z menu Picaxe, nebo stiskem F8.

Například:

```
main:     for b1 = 0 to 63                    ' start a loop
          sertxd("The value of b1 is ",#b1,13,10)
          pause 1000
          next b1                             ' next loop
```

servo

syntaxe:

SERVO pin,pulse

- pin je proměnná nebo konstanta (0-7) určující, který vývod bude použit.

- pulse je proměnná nebo konstanta (75-225) určující polohu serva (šířku pulsu)

funkce:

Příkaz servo posílá na určený výstup řídící pulsy s opakovací periodou 20 ms, které mohou sloužit k ovládání modelářského servomechanismu (serva). Pulsy jsou vysílány trvale i po provedení příkazu.

Serva, často používaná modeláři, vyžadují ke svému řízení impulsy o šířce 1 až 2 ms opakované neustále každých 20 ms.

Některá serva fungují v širším rozsahu řídících pulsů až od 0,5 do 2,5 ms. Tento širší rozsah je třeba vyzkoušet opatrně, aby nedošlo ke zničení převodů, kdyby se servo snažilo překonat mechanický doraz.

Například:

```
main:      servo 4,75          ' move servo to one end
           pause 2000          ' wait 2 seconds
           servo 4,150         ' move servo to centre
           pause 2000          ' wait 2 seconds
           servo 4,225         ' move servo to other end
           pause 2000          ' wait 2 seconds
           goto main           ' loop back to start
```

setint

syntaxe:

SETINT input,mask

- input je proměnná nebo konstanta (0-255), která udává hodnoty vstupů

- mask je proměnná nebo konstanta (0-255), která určuje masku – použité vstupy.

funkce:

Umožní přerušit běh programu při zadané kombinaci vstupů – zpracování asynchronních událostí.
Podrobnosti naleznete v originální anglické příručce.

setfreq

syntaxe:

SETFREQ freq

- freq je klíčové slovo m4 nebo m8.

funkce:

Nastaví hodinovou frekvenci u kontrolerů s interním oscilátorem na 4MHz nebo 8MHz. Obvyklá frekvence nastavená po zapnutí napájení je 4MHz. Vyšší frekvence umožňuje rychlejší běh programu, ale má vliv na řadu příkazů, které závisí nějakým způsobem na časování.

Například:

```
setfreq m4              ' setfreq to 4 MHz
readtemp 1,b1           ' do command at 4 MHz
setfreq m8              ' set freq back to 8 MHz
```

shiftin (spiin)

syntaxe:

SPIIN hodiny,sdata,režim,(proměnná {/ bitů} {, proměnná {/ bitů}, …})

- hodiny je proměnná nebo konstanta (0-7) udávající, který pin má být použit jako hodiny.

- sdata je proměnná nebo konstanta (0-7) udávající, který pin má být použit pro data.

- režim je proměnná nebo konstanta (0-7) udávající režim:??????????????
0 LSBPre_L (LSB první, vzorek před hodinami, prodlevy v log. 0)
1 MSBPre_L (MSB první, vzorek před hodinami, prodlevy v log. 0)
2 LSBPost_L (LSB první, vzorek za hodinami, prodlevy v log. 0)
3 MSBPost_L (MSB první, vzorek za hodinami, prodlevy v log. 0)
4 LSBPre _H (LSB první, vzorek před hodinami, prodlevy v log. 1)

5 MSBPre_H (MSB první, vzorek před hodinami, prodlevy v log. 1)
6 LSBPost_H (LSB první, vzorek za hodinami, prodlevy v log. 1)
7 MSBPost _H (MSB první, vzorek za hodinami, prodlevy v log. 1)

- proměnná proměnná pro uložení dat.

- bity volitelný počet bitů pro přenos. Implicitně 8.

Příkaz spiin (kompilátor akceptuje i shiftin) je „bit-bang“??????????? metodou komunikace SPI na X1 a X2. Hardwarové řešení viz příkaz „hshin“.

Do proměnné je umístěno implicitně 8 bitů. Volitelně lze zadat počet bitů (1 až 8). Takže když například chcete přijmout 12 bitů, rozdělte je do dvou bajtů, do jednoho přijměte 8 bitů a do druhého 4 bity. Pamatujte si, že pokud přenášíte nejprve LSB, bity jsou posunuty doprava, takže posunutí pouze 4 bitů by je ponechalo v bitech 7-4 (ne 3-0). Přenášíte-li nejprve MSB, bity jsou posunuty doleva.

Připojujete-li zařízení SPI (např. EEPROM), pamatujte si, že datový vstup paměti EEPROM musíte připojit na datový výstup PICAXE a naopak.

Jiné mikrokontroléry PICAXE nemají přímý příkaz spiin (shiftin).

Stejné funkce můžete samozřejmě používat i v jiných produktech pomocí podprogramů uvedených na druhé straně.

Vliv zvyšování rychlosti hodin:

Zvyšování rychlosti hodin zvyšuje hodinový kmitočet SPI.

Například:

```
spiin 2,1,LSB_Pre_H, (b1 / 8)      ' přenos 8 bitů do b1
```

shiftin/shiftout na čipech PICAXE bez nativních příkazů:

Některé mikrokontroléry PICAXE nemají příkaz shiftin. Stejné funkce lze v jiných produktech dosáhnout použitím níže uvedených podprogramů. Tyto podprogramy jsou také uloženy v souboru shiftin_out.bas inv adresáři \samples software Programming Editor.

Chcete-li je použít, jednoduše zkopírujte definice symbolů na začátek vašeho programu a zkopírujte odpovídající podprogramy shiftin na konec vašeho programu.

Nekopírujte všechno, protože to zbytečně zabírá paměť.

Předpokládá se, že oba výstupy, datový i hodinový, jsou před použitím podprogramu v logické 0.

Řádek BASICu

“shiftin sclk, sdata,mode, (var_in(\bits)) “

se změní na

gosub shiftin_LSB_Pre (pro režim LSBPre)

gosub shiftin_MSB_Pre (pro režim MSBPre)

gosub shiftin_LSB_Post (pro režim LSBPost)

gosub shiftin_MSB_Post (pro režim MSBPost) ‘

```
' ~~~~~ SYMBOL DEFINITIONS ~~~~~
' Vyžadováno pro všechny rutiny. Podle potřeby změňte čísla pinů.
' Používá proměnné b7-b13 (tj. b7,w4,w5,w6). Pokud používáte pouze 8 bitů
' mohou být všechny proměnné word bezpečně změněny na byte.
'
'***** Vzorová definice symbolů *****
symbol sclk = 5              ' hodiny (výstupní pin)
symbol sdata = 7             ' data (výstupní pin pro shiftout)
symbol serdata = input7      ' data (vstupní pin pro shiftin, viz input7
symbol counter = b7          ' proměnná použitá v cyklu
symbol mask = w4             ' proměnná pro maskování bitů
symbol var_in = w5           ' proměnná pro data použitá v průběhu shiftin
symbol var_out = w6          ' proměnná pro data použitá v průběhu shiftout
```

```
symbol bits = 8              ' počet bitů
symbol MSBvalue = 128        ' MSBvalue
'
'(=128 pro 8 bitů, 512 pro 10 bitů, 2048 pro 12 bitů)

'=============================================================================
' ~~~~~ RUTINY SHIFTIN ~~~~~
' Pouze jediná z těchto 4 je povinná - viz vaše požadavky IC
' Doporučujeme ostatní smazat a ušetřit tak místo
'=============================================================================
' ***** LSB první, Data Pre-Clock *****
shiftin_LSB_Pre:
let var_in = 0
for counter = 1 to bits           ' počet bitů
var_in = var_in / 2               ' posun doprava, aby byl LSB první
if serdata = 0 then skipLSBPre
var_in = var_in + MSBValue        ' nastavení MSB je-li serdata = 1
skipLSBPre: pulsout sclk,1        ' změna hodin pro načtení dalšího bitu dat
next counter
return
'=============================================================================
' ***** MSB první, Data Pre-Clock *****
shiftin_MSB_Pre:
let var_in = 0
for counter = 1 to bits           ' počet bitů
var_in = var_in * 2               ' posun doleva, aby byl MSB první
if serdata = 0 then skipMSBPre
var_in = var_in + 1               ' nastavení LSB je-li serdata = 1
skipMSBPre: pulsout sclk,1        ' změna hodin pro načtení dalšího bitu dat
next counter
return
'=============================================================================
' ***** LSB první, Data Post-Clock ***** '
shiftin_LSB_Post: let var_in = 0
for counter = 1 to bits           ' počet bitů
var_in = var_in / 2               ' posun doprava, aby byl LSB první
pulsout sclk,1                    ' změna hodin pro načtení dalšího bitu dat
if serdata = 0 then skipLSBPost
var_in = var_in + MSBValue        ' nastavení MSB je-li serdata = 1
skipLSBPost: next counter
return
'=============================================================================
' ***** MSB první, Data Post-Clock *****
shiftin_MSB_Post: let var_in = 0
for counter = 1 to bits           ' počet bitů
var_in = var_in * 2               ' posun doleva, aby byl MSB první
pulsout sclk,1                    ' změna hodin pro načtení dalšího bitu dat
if serdata = 0 then skipMSBPost
var_in = var_in + 1               ' nastavení LSB je-li serdata = 1
skipMSBPost: next counter
return
'=============================================================================
```

shiftout (spiout)

syntaxe:

SPIOUT hodiny,sdata,režim,(data{/ bitů}, {data{/ bitů},…})

- **hodiny** je proměnná nebo konstanta (0-7) udávající, který pin má být použit jako hodiny.
- **sdata** je proměnná nebo konstanta (0-7) udávající, který pin má být použit pro data.
- **režim** je proměnná nebo konstanta (0-3) udávající režim:
  - **0** LSBFirst_L (LSB první, prodlevy v log. 0)
  - **1** MSBFirst_L (MSB první, prodlevy v log. 0)
  - **4** LSBFirst_H (LSB první, prodlevy v log. 1)
  - **5** MSBFirst_H (MSB první, prodlevy v log. 1)
- **data** je proměnná nebo konstanta (0-7) obsahující data k odeslání.
- **bitů** (volitelně) je počet bitů, které mají být odeslány. Pokud jejich počet nebude uveden, implicitně se použije 8.

Příkaz spiout (kompilátor akceptuje i shiftout) je bit-bang?????? metodou komunikace SPI na X1 a X2. Hardwarové řešení viz příkaz „hspiout“.

Implicitně se přenáší 8 bitů. Volitelně můžete nadefinovat jiný počet bitů (1 až 8). Potřebujete-li například přenést 12 bitů, udělejte to ve dvou bajtech, v jednom přeneste 8 bitů a ve druhém 4. Pamatujte si, že pokud používáte metodu „MSB jako první“, bity jsou posunuty doleva, takže když přenášíte pouze 4 bity, musí být uloženy v bitech 7-4 (ne 3-0). Při metodě LSB jsou bity posunuty zprava.

Připojujete-li zařízení SPI (např. EEPROM), pamatujte si, že datový vstup paměti EEPROM musíte připojit na datový výstup PICAXE a naopak.

Některé mikrokontroléry PICAXE nemají vestavěný příkaz shiftout. Stejné funkce lze v jiných produktech dosáhnout použitím níže uvedených podprogramů.

Vliv zvyšování rychlosti hodin:

Zvyšování rychlosti hodin zvyšuje hodinový kmitočet SPI.

Například:

```
spiout 1,2,LSB_First, (b1 / 8)     ' odeslání 8 bitů z b1
```

shiftin/shiftout na čipech PICAXE bez nativních příkazů:

Některé mikrokontroléry PICAXE nemají příkaz shiftout. Stejné funkce lze v jiných produktech dosáhnout použitím níže uvedených podprogramů. Tyto podprogramy jsou také uloženy v souboru shiftin_out.bas inv adresáři \samples software Programming Editor.

Chcete-li je použít, jednoduše zkopírujte definice symbolů na začátek vašeho programu a zkopírujte odpovídající podprogramy shiftin na konec vašeho programu.

Nekopírujte všechno, protože to zbytečně zabírá paměť.

Předpokládá se, že oba výstupy, datový i hodinový, jsou před použitím podprogramu v logické 0.

Řádek BASICu

„shiftout sclk, sdata,mode, (var_out(\bits))“

se změní na

gosub shiftout_LSBFirst (pro režim LSBFirst)

gosub shiftout_MSBFirst (pro režim MSBFirst)

Pamatujte si, že definice symbolů uvedených u příkazu „shiftin“ musí být také použity.

```
'=============================================================================
' ***** Odeslání LSB jako prvního *****
shiftout_LSBFirst:
for counter = 1 to bits      ' počet bitů
mask = var_out & 1           ' maskování LSB
low sdata                    ' data do log. 0
```

```
if mask = 0 then skipLSB
high sdata                    ' data do log. 1
skipLSB: pulsout sclk,1       ' hodinový impuls 10us
var_out = var_out / 2         ' posunutí proměnné doprava pro LSB
next counter
return
'================================================================================
' ***** Odeslání MSB jako prvního *****
shiftout_MSBFirst:
for counter = 1 to bits       ' počet bitů
mask = var_out & MSBValue     ' maskování MSB
low sdata                     ' data do log. 0
if mask = 0 then skipMSB
high sdata                    ' data do log. 1
skipMSB: pulsout sclk,1       ' hodinový impuls 10us
var_out = var_out * 2         ' posunutí proměnné doleva pro MSB
next counter
return
'================================================================================
```

sleep

syntaxe:

SLEEP period

- period je proměnná nebo konstanta (0-65535), která udává dobu uspání kontroleru v násobcích 2,3 s.

funkce:

Tento příkaz uspí kontroler na zadaný počet násobků 2,3 s. Příkaz využívá watchdog timer s omezenou přesností časování.

Například:

```
main:       high 1        ' switch on output 1
            sleep 10      ' sleep for 23 seconds
            low 1         ' switch off output 1
            sleep 100     ' sleep for 230 seconds
            goto main     ' loop back to start
```

sound

syntaxe:

SOUND pin,(note,duration,note,duration…)

- pin je proměnná nebo konstanta (0-7), určující, který vývod bude použit.

- note jsou proměnné nebo konstanty (0-255), které určují typ a frekvenci tónu. 0 značí pauzu, hodnoty 1-127 jsou tóny stoupající frekvence, 128-255 je bílý šum stoupající frekvence.

- duration jsou proměnné nebo konstanty (0-255), které určují délku tónu v násobcích 10 ms.

funkce:

Tento příkaz vytváří pípnutí vhodné například k upozornění obsluhy, indikaci stisku tlačítka apod. Ke generování hudebního signálu jsou určeny příkazy play nebo tune. Parametry note a duration musí být uvedeny vždy v páru.

Například:

```
main:       let b0 = b0 + 1   ' increment b0
            sound 7,(b0,50)   ' make a sound
            goto main         ' loop back to start
```

stop

syntaxe:
STOP
funkce:
Ukončí běh programu. K obnovení funkce mikrokontroléru dojde vypnutím a opětovným zapnutím napájení nebo zavedením nového programu z PC
Příkaz stop ukončí program. Na rozdíl od příkazu end se mikrokontrolér nenastaví do úsporného režimu, takže příkazy servo a pwmout pokračují v činnosti.

Například:

```
main:       pwmout 1,120,400
            stop
```

switch on/off

syntaxe:
SWITCH ON pin
SWITCHON pin
SWITCH OFF pin
SWITCHOFF pin
- pin je proměnná nebo konstanta, označující i/o pin

funkce:
Nastaví pin na high / low

Pro informaci:
Tento příkaz je vytvořen pro nejmladší uživatele PICAXE a je zcela ekvivalentní příkazům „high“ a „low“.

Například:

```
main:       switch on 7         ' switch on output 7
            wait 5              ' wait 5 seconds
            switch off 7        ' switch off output 7
            wait 5              ' wait 5 seconds
            goto main           ' loop back to start
```

symbol

syntaxe:
SYMBOL názevsymbolu = hodnota
SYMBOL názevsymbolu = hodnota ?? konstanta
- **názevsymbolu** je textový řetězec, který musí začínat písmenem nebo „_“.
  Za prvním znakem mohou následovat i čísla (0 - 9).
- **hodnota** je proměnná nebo konstanta, které bude přiřazen alternativní název symbolu.
- **??** libovolná podporovaná matematická funkce, např. + - * / atd.

funkce:
Přiřazení hodnoty novému symbolu. U konstant mohou být použity i matematické operátory (ne u proměnných)

Symboly se používají k pojmenování konstant nebo proměnných, aby se snáze pamatovaly a aby byl zřejmější jejich účel. Symboly nemají vliv na délku programu, jelikož jsou před stažením převedeny zpátky na „čísla“.

Názvy symbolů mohou obsahovat znaky čísel, ale nesmí začínat číslem. Názvy symbolů se nesmí shodovat s názvy příkazů nebo vyhrazenými slovy, jako napr. input, step, atd. Viz seznam vyhrazených slov na konci této sekce.

Při definování vstupních a výstupních proměnných dávejte pozor, abyste zadali „pin0“ a ne „0“ při zadávání vstupních proměnných, které mají být použity v příkazech if…then.

Například:

```
symbol RED_LED = 7          ' definice výstupního pinu
symbol PUSH_SW = pin1       ' definice vstupního tlačítka
symbol DELAY = B0                ' definice symbolu proměnné
let DELAY = 200             ' předvyplnění čítače číslem 200
main: high RED_LED          ' zapnutí výstupu 7
pause DELAY                      ' čekání 0,2 sekundy
low RED_LED                      ' vypnutí výstupu 7
pause DELAY                      ' čekání 0,2 sekundy
goto main                   ' zpátky na začátek
```

toggle

syntaxe:

TOGGLE pin

- pin je proměnná nebo konstanta (0-7) určující, který vývod bude použit.

funkce:

Změní logickou úroveň na určeném výstupu.

Například:

```
main:     toggle 7    ' toggle output 7
          pause 1000  ' wait 1 second
          goto main   ' loop back to start
```

tune

syntaxe:

TUNE LED, speed, (note, note, note…) (platí pouze pro PICAXE-08M)

TUNE pin, speed, (note, note, note…) (platí pouze pro PICAXE-14M)

- LED je proměnná nebo konstanta (0-3) určující způsob blikání připojených LED během hraní.
0 - bez blikání
1 - výstup 0 se zapíná a vypíná
2 - výstup 4 se zapíná a vypíná
3 - výstupy 0 a 4 se střídavě zapínají a vypínají

- pin je proměnná nebo konstanta (0-7) určující, který vývod bude použit pro výstupní signál. Nelze použít pro PICAXE-08M, kde je výstupní signál pevně nastaven na výstup 2.

- speed je proměnná nebo konstanta (1-15) určující tempo přehrávání skladby.

- note jsou konstanty (data), generovaná pomocí programu Tune Wizard.

Podrobnosti naleznete v originální anglické příručce.

wait

syntaxe:

WAIT seconds

- seconds je konstanta, která určuje dobu čekání v sekundách

funkce:

Tento příkaz je ekvivalentní ‚pause * 1000‘, jedná se o jakési makro a nemůže být z tohoto důvodu použit s proměnnou.

Například:

```
main:      switch on 7        ' switch on output 7
           wait 5             ' wait 5 seconds
           switch off 7       ' switch off output 7
           wait 5             ' wait 5 seconds
           goto main          ' loop back to start
```

write

syntaxe:

WRITE location,data

- location je proměnná nebo konstanta určující a adresu (0-255).

- data je proměnná nebo konstanta představující data určená k zápisu.

funkce:

Příkaz zapisuje data do interní eeprom mikrokontroleru. Obsah této paměti se uchová i po vypnutí napájení. Tato data jsou zapisována při každém zavedení nového programu do mikrokontroléru, podle definice v příkazu DATA/EEPROM. Za běhu programu mohou být tato data čtena pomocí příkazu read. Příkaz WRITE pracuje pouze s 8 bitovými daty. Pokud je třeba zapsat proměnnou word, dosáhne se toho použitím dvou příkazů WRITE s oběma 8bitovými proměnnými: (například w0 se zapíše tak, že se zapíše b0 a b1). Velikost, umístění a případná omezení použitelnosti eeprom u jednotlivých typů kontrolérů jsou uvedeny u příkazu DATA.

Například:

```
main:      for b0 = 0 to 63   ' start a loop
           serin 6,T2400,b1   ' receive serial value
           write b0,b1        ' write value into b1
           next b0            ' next loop
```