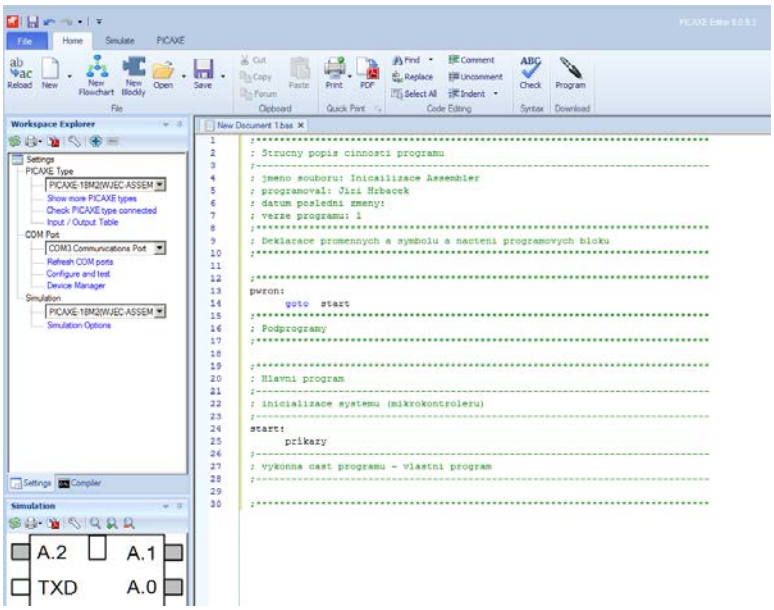
Programování PICAXE18M2 v Assembleru

Nastavení programming editoru

PICAXE PROGRAMMING EDITOR 6

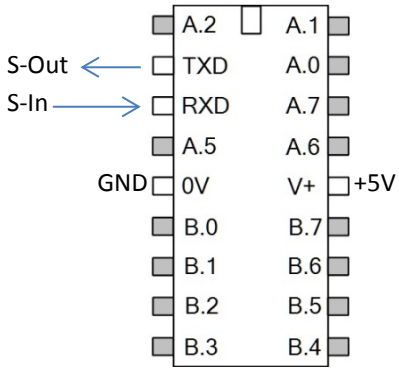
Nastavit PICAXE Type – PICAXE – 18M2(WJEC-ASSEMBLER, stejně tak nastavit Simulation

Pokud tam není, otevřeme přes File – Options okno nastavení. Zde vybereme položku Compiler a zde v části Display following special PICAXE types vybereme PICAXE-18M2(WJEC-ASSEMBLER).



Připojení PICAXE18M2 k robotovi pro programování

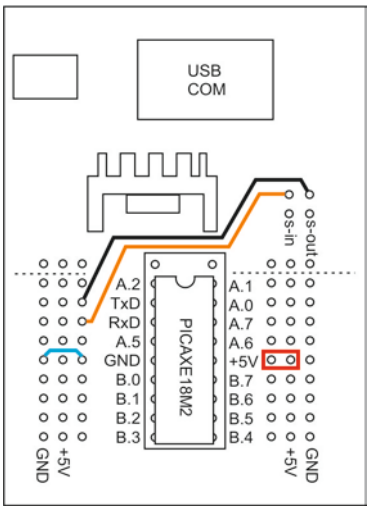
Zapojení S-OUT na TxD pin2 PICAXE18M2 a S-IN na RxD pin 3 PICAXE18M2 (obrázek vlevo).



Osazení PICAXE 18M2 do desky procesoru robota HSES

PICAXE 18M2 vložíme do patice desky procesoru do dolní části. PIN 1 a 20 zůstane prázdný!!!! Na tyto piny je na desce procesoru přivedeno napájecí napětí GND a +5V.

Musíme vyjmout propojky S-Out a S-In a tyto piny dvojvodičem přivést na TxD a RxD procesoru. Stejně tak propojíme GND k pinu 5 procesoru pomocí propojovacího vodiče. K pinu 14 procesoru připojíme +5V například pomocí jedné vyjmuté propojky.



PICAXE18M2 je postaven na mikrokontroleru PIC16F1847. Chceme-li se na něm učit programovat v assembleru, využijeme WJEC Assembler ve spojení s PICAXE18M2, což plnohotně emuluje (chová se stejně, jako by skutečně byl) všechny instrukce mikrokontroler PIC16F88 (bohužel zcela, neemuluje všechny registry a tím ani všechny vnitřní systémy). Studentská referenční příručka pro WJEC Assembler je zde: http://www.picaxe.com/docs/wjec_instruction_set.pdf.

Pro tuto chvíli je to pro naši výuku plně dostačující. Při emulaci není rychlost provádění instrukcí stejná, jako bychom mikrokontrolér PIC16F88 přímo naprogramovali v assembleru. Později budeme mikrokontrolery programovat přímo v assembleru. Budou pak schopny provádět až desítky milionů instrukcí za sekundu a plně využívat všechny své vnitřní systémy. Tím vším se budeme podrobně zabývat později.

Pokud píšeme programy v assembleru, je obvyklé, že jejich soubory mají příponu ASM. Pokud budeme programovat PICAXE18M2 v assembleru pomocí PICAXE PROGRAMMING EDITORU 6 a překladače WJEC ASSEMBLER, musí mít soubor s programem příponu BAS.

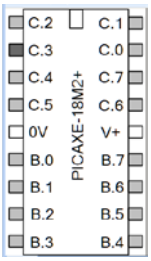
Je dobré, aby měl program jasnou strukturu. Lépe se v něm pak orientujeme a předejdeme vzniku nepředvídaných chyb. Doporučená struktura programu je následující:

```
;*******************************************************************************
; Strucny popis cinnosti programu
;-------------------------------------------------------------------------------
; jmeno souboru:
; programoval:
; datum posledni zmeny:
; verze programu:
;*******************************************************************************
; Deklarace promennych a symbolu a nacteni programovych bloku
;*******************************************************************************

;*******************************************************************************
pwron:    goto      start
;*******************************************************************************
; Podprogramy
;*******************************************************************************

;*******************************************************************************
; Hlavni program
;-------------------------------------------------------------------------------
; inicializace systemu (mikrokontroleru)
;-------------------------------------------------------------------------------
start:
          instrukce1
          instrukce2

;-------------------------------------------------------------------------------
; vykonna cast programu - vlastni program
;-------------------------------------------------------------------------------

;*******************************************************************************
```

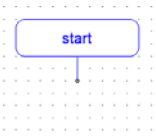
Porovnání Basicu a Assembleru

Po zapnutí napájení jsou všechny IO piny nastaveny jako vstupy

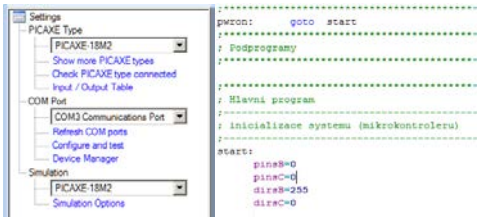
Basic



V Basicu je Port B a Port C na pinech jak je na tomto obrázku.

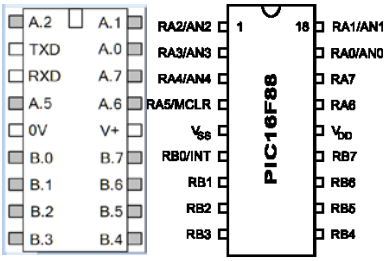


Ve Flowchartu je automaticky nastaven Port B jako výstupy a Port C jako vstupy



V Basicu použijeme příkazy pro nastavení stavu výstupů příkazy pinsC a pinsB. 1 v odpovídajícím bitu nastavuje výstup, 0 v odpovídajícím bitu nastaveje vstup.

Assembler



Obsazení pinů – levý a střední obrázek se váže k WJEC Assembleru a programování PICAXE18M2 v assembleru. Vpravo pak originální z datového listu Microchip.

Seznam instrukcí WJEC Assembleru PICAXE18M2

WJEC Assembler neemuluje plně mikrokontrolér PIC16F88. Emuluje jeho instrukce, ale neemuluje všechny registry. Proto se na něm začneme assembler učit, později však přejdeme na PIC16F88 a budeme jej programovat přímo, ne přes PICAXE PROGRAMMING EDITOR.

Mnemonic	Operands	Description
addlw	k	Add working register to literal k (k + WREG)
andlw	k	AND working register with literal k (k & WREG)
bcf	f, b	Bit clear in file register (file register f, bit b)
bsf	f, b	Bit set in file register (file register f, bit b)
btfsc	f, b	Bit test in file register, skip if clear (file register f, bit b)
btfss	f, b	Bit test in file register, skip if set (file register f, bit b)
call	label	Call subroutine at label
clrf	f	Clear file register f
comf	f, d	Complement f (to itself if d = 1, or working register if d = 0)
decfsz	f, d	Decrement f, skip if zero (to itself if d = 1, or working register if d = 0)
goto	label	Unconditional branch to label
incf	f, d	Increment file register f (to itself if d = 1, or working register if d = 0)
iorlw	k	Inclusive OR working register with literal (k | WREG)
movf	f, d	Move f (to itself if d = 1, or working register if d = 0)
movlw	k	Move literal to working register
movwf	f	Move working register to file register f
nop	-	No operation
retfie	-	Return from interrupt service routine and set global interrupt enable bit GIE
return	-	Return from subroutine
sublw	k	Subtract working register from literal k (k – WREG)

Ve WJEC Assembleru budeme používat následující předdefinované:

speciální registry

PORTA, PORTB	obousměrný vstupně/výstupní port
TRISA, TRISB	řídící registry portů (volí, který pin portu bude jako vstup a který jako výstup)
STATUS	stavový registr procesoru
INTCON	řídící registr přerušení

Poznámka: *při práci s registry PORTA a PORTB musí být nastaven bit STATUS registru RP0=0, při práci s registry TRISA a TRISB musí být nastaven bit STATUS registru RP0=1, při práci s registry STATUS a INTCON nezáleží na nastavení bitu RP0 STATUS registru.*

univerzální registry

Tyto registry jsou přístupné pod názvem B0 až B27. Při práci s nimi je třeba aby byl bit RP1 i RP0 v 0.

Můžeme je však přejmenovat pomocí definice symbolů, například

```
redLED      EQU   d'1'
wSave       EQU   B20
```

předdefinované konstanty

pro použití v instrukcích

W	výsledek uložit do W registru (hodnota 0)
F	výsledek uložit do registru (hodnota 1)

registr STATUS

C	carry bit – přenos
Z	zero bit – výsledek operace je 0
RP0	page selection bit (při přímém adresování 8. bit adresy registru)

registr INTCON

GIE	global interrupt enable
INT0IE	povolení přerušení od vstupu RB0
INT0IF	příznak – došlo k vnějšímu přerušení ze vstupu RB0 (je třeba nulovat programem)

Používání zápisu čísel

d’123’	decimal – desítková čísla (default)
b’10101010’	binary – dvojková čísla
0b10101010	
h’FF’	hex – čísla v šestnáctkové soustavě
0xFF	

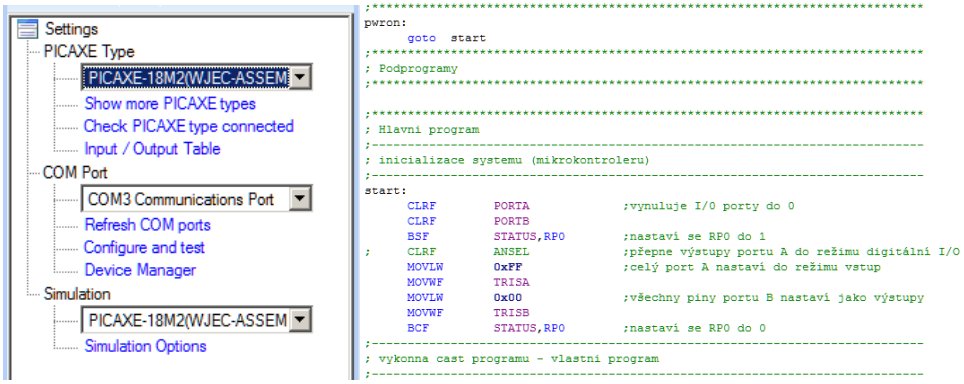
Předdefinované podprogramy

wait1ms	čekej 1 millisekundu
wait10ms	čekej 10 millisekund
wait100ms	čekej 100 millisekund
wait1000ms	čekej 1000 millisekund
readadc0	přečti analogovou hodnotu na A.0 a ulož ji do b0
readadc1	přečti analogovou hodnotu na A.1 a ulož ji do b1
readadc2	přečti analogovou hodnotu na A.2 a ulož ji do b2
readtemp0	přečti teplotu z DS18B20 připojeného na A.0 a ulož ji do b0
readtemp1	přečti teplotu z DS18B20 připojeného na A.1 a ulož ji do b1
readtemp2	přečti teplotu z DS18B20 připojeného na A.2 a ulož ji do b2

Práce s I/O porty

Inicializace portů v assembleru

Na následujícím obrázku je inicializace mikrokontroléru v WJEC Assembleru.



Porovnání inicializace portů v Basicu a WJEC Assembleru

Basic	Assembler
dirsB = 255 dirsC = 0	bsf STATUS,RP0 movlw 0xff movwf TRISB movlw 0x00 movwf TRISA bcf STATUS, RP0
pinsC = 0 pinsB = 0	clrf PORTA clrf PORTB

Popis použitých instrukcí

CLRF	Clear f	Vynuluj registr

CLRF *f*

00h do (f) f je adresa registru (dolních 7 bitů adresy, horní 2 bity jsou v bitech RP1 a RP0 registru STATUS)
STATUS (bit 2) Z bit nastaví do 1

BSF	Bit Set f	Nastav bit registru

BSF *f*,*b*

1 do (f<b>) nastaví jedničku do bitu b (bit 0 až 7) registru na adrese f (dolních 7 bitů adresy, horní 2 bity jsou v bitech RP1 a RP0 registru 03h - STATUS)

BCF	Bit Clear f	Vynuluj bit registru

BCF *f*,*b*

1 do (f<b>) nastaví nulu do bitu b (bit 0 až 7) registru na adrese f (dolních 7 bitů adresy, horní 2 bity jsou v bitech RP1 a RP0 registru 03h - STATUS)

MOVLW	Move Literal to W	Ulož konstantu do W registru

MOVLW *k*

k do (W) uloží konstantu do W registru
k – konstanta 00h až FFh

MOVWF	Move W to F	Ulož obsah registru W do registru o adrese F

MOVWF *f*

(W) do (f) přesune (zkopíruje) obsah W registru do registru f (dolních 7 bitů adresy, horní 2 bity jsou v bitech RP1 a RP0 registru 03h - STATUS)