Řada MPC400
programovací příručka

Řada MPC400 – programovací příručka

Přístup k I/O

Funkce a používání grafického zobrazení

Vestavěné funkce pro SIMPLE4

Komunikační možnosti

2. verze dokumentu,

OBSAH

1. KOMPATIBILITA 8
2. VSTUPY / VÝSTUPY 9
Číslování uzlů 9
Vstupy/výstupy ve zdrojovém textu SIMPLE4 9
Digitální vstupy/výstupy 10
Analogové vstupy/výstupy 10
Ostatní registry zařízení 11
I/O moduly pro řadu 400 11
- modul A 11
- modul B 12
- modul C 12
- modul D 14
- modul E 14
- modul F 15
- modul G 15
- modul H 16
- modul I 16
- modul J 17
- modul K 17
- modul L 18
- modul M 18

Bezpečnostní funkce 19
3. ZOBRAZOVACÍ SYSTÉM 20
Vrstvy 20
- Vrstva TEXT 20
- Vrstva GRAPHIC 21
- Vrstva BACKGROUND 22

Barvy 22
Bitmapy 24
Chod zobrazování 24
4. VESTAVĚNÉ FUNKCE 26
Časovací funkce 26
- GetTickCount 26

Systémové funkce 26
- SYS_GetPLCInfo 26
- SYS_ExbusReset 29

- SYS_PLCReset 29

Funkce pro převod hodnoty odporu na teplotu 29

- R2Pt100 30
- R2Pt1000 30
- R2Ni1k5000 31
- R2Ni1k6180 31

Matematické funkce 31

- Deg2Rad 32
- Rad2Deg 32
- Sqrt 32
- Sin 33
- Cos 33
- Tan 33
- ArcSin 33
- ArcCos 34
- ArcTan 34
- Ln 34
- Pow 34
- Exp 35
- Ceil 35
- Floor 35
- FMod 36
- ModF 36

Funkce pro indexový přístup k I/O 36

- IOReadX 38
- IOWriteX 38
- IOReadY 38
- IOWriteY 39
- IOReadI 39
- IOWriteI 40
- IOReadO 40
- IOWriteO 40

Doplňkové funkce I/O subsystému 41

- IOGetNodeDef 41
- IODefVirtualNode 41
- IOList, IOVirt 44

Funkce vrstvy TEXT - textová obrazovka 44

- Display 46
- Display 46
- DisplayClear 47
- DisplaySize 48

- DisplayFColor 48
- DisplayBColor 48
- DisplayGetFColor 49
- DisplayGetBColor 49
- DisplaySetFont 49
- DisplaySetSize 49
- DisplaySetOrg 50
- DisplayCursorCode 50
- DisplayEnable 51
- IsDisplayEnable 51
- DisplayGetLine 51

Vrstva GRAPHIC 51

- GdiFunctionTime 52
- GdiThreadTime 52
- GdiGraphSize 52
- GdiWindow 53

GRAPHIC - práce s barvami 53

- GdiSetUserColor 53
- GdiFColor 54
- GdiBColor 54
- GdiGetFColor 54
- GdiGetBColor 54
- GdiFillScr 55

GRAPHIC - práce s textem 55

- GdiSetFont 55
- GdiTextRect 57
- GdiTextMode 57
- GdiText 59
- GdiDisplayText 59
- GdiGetTextLen 60
- GdiGetTextHeight 60

GRAPHIC - práce s vektorovou grafikou 61

- GdiPenType 61
- GdiPoint 62
- GdiLine 62
- GdiBeginPoly 62
- GdiPoly 62
- GdiFillPoly 63
- GdiNAngle 63
- GdiFillNAngle 64
- GdiRect 64

- GdiFillRect 64
- GdiCirc 65
- GdiFillCirc 65
- GdiArc 66
- GdiSlice 66
- GdiFillSlice 67
- GdiBMP 67

Vrstva BACKGROUND 67

- GdiBkgSetColor 68
- GdiBkgBMP 68

Komunikace přes linku RS485 69

- UartConfig 69
- UartTxR 69
- UartStopTxR 70
- CRC16_RS 70
- CRC16_LS 70

Zpracování textu a dat 71

- Copy 71
- Fill 74
- StrLen 74
- StrIns 74
- StrRep 75
- StrDel 76

Datové přenosy 77

- TrResetBuffer 77
- TrWriteBuffer 77
- TrReadBuffer 79
- TrGetBufferGap 81
- TrGetBufferDataLen 81
- TrCompare 81
- TrSetParam 82
- TrScanBuffer 83
- TrInitSymbol 84
- TrFindDataRead 85

5. DATOVÉ STRUKTURY 87

- rtc_type 87
- pxy 88
- bmp 89
- bmpmode 89
- font 91
- linept 92

- nangle 92
- arcmode 92
- _uart 93
- _uart_crc 96
- dataptr 96
- _circular_buffer 98
- _scan_value 98
- _scan_symbol 99
- _wr_buffer 99
- _rd_buffer 100
- PNET_STATUS, SPNET_STATUS 101

1. KOMPATIBILITA

Řídicí systémy MICROPEL řady 400 jsou logickým pokračováním předchozích typů PLC MICROPEL. Jsou zde navíc nové možnosti a funkce, původní funkce a vlastnosti PLC MPC300, MPC200, K1, K10 však zůstávají. Popis programovací příručky je společný pro všechny PLC řady MPC400 a všechny další PLC, které jsou technologicky z této řady odvozeny tj. MT424, MT470, MCA5xx atd. a pro následující čísla firmware a vyšší:

MPC400 – 5056, MPC404 – 5810, MPC41x – 6003, MT470 – 5416, MT424 – 5635,

MCA5xx –1609, MCA6xx – 1408, MEX40x – 5256.

Jedinou hlubší změnou, která vyžaduje úpravy v uživatelských programech, je odlišný přístup ke vstupům/výstupům. Souvisí to s novou koncepcí stavby rozsáhlejších systémů z komponentů řady 400. Detailní popis viz kap. VSTUPY / VÝSTUPY.

Kompatibilní funkce a vlastnosti

Všechny následující možnosti a funkce zůstávají stejné jako u MPC300:

STACK Zásobník, 11776x WORD, přístup přes registr POINTER, není-li uvedeno jinak

T0 ...T7 Časovače, jejich funkce, frekvence i sady řídicích bitů TEN, TPA, TDM, TOE

RESET Bit, pouze nastavený po restartu PLC, nulovaný uživatelem

DISPLAY Funkce pro tisk textů a hodnot proměnných na displej PLC

FORMAT Škála různých formátů pro tisk hodnot a práci s uživatelskými znaky

RTC Reálný čas s registry: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, WEEK.

KBCODE Kód stisknuté klávesy

Nové funkce, vlastnosti a rozšíření

Funkce a vlastnosti, které proti MPC300 doznaly vylepšení:

Program Velikost paměti pro uživatelský program je 768 kB.

Data Paměť pro uživatelská data je 84 kB, s rozšířením /4M je to 468 kB.

Rychlost Typická rychlost běhu programu je cca 200-1000 průchodů za sekundu.

I/O Přímé připojení až 3800 vstupů/výstupů speciální periferní linkou až 460 kbit/s.

Komunikace 2x RS485 s možností alternativních funkcí (EXbus, uživ. UART, MODBUS...),
USB pro přímé připojení k PC a rychlejší ladění i zatahování programů.

Displej Barevný, text i grafika, textově: až 16x40 znaků, graficky: 800x480 bodů.

Grafika Vestavěné funkce pro grafiku na displeji.

Matematika Vestavěné matematické a goniometrické funkce.

Uvedená vylepšení platí pro všechny PLC řady 400 a technologicky odvozené typy, pokud není ve specifikaci výrobku uvedeno jinak.

2. VSTUPY / VÝSTUPY

Z pohledu programátora není u řady 400 rozdíl mezi vstupy/výstupy umístěnými na centrální jednotce a vstupy/výstupy na periferní lince EXbus. Vstupy/výstupy jsou strukturovaně umístěny v poli s kapacitou cca 256 I/O uzlů. Jeden uzel může obsahovat různé množství vstupů/výstupů - od 16 analogových až po cca 64 digitálních. Prvních 16 uzlů je vyhrazeno pro vlastní PLC - první 3 uzly (0..2) zabírají volitelné I/O moduly, ostatní uzly jsou vyhrazeny pro různé virtuální systémové služby. Zbylých 240 uzlů může být přiřazeno teoreticky až 240 zařízením na EXbus. Jednotky MEX400 zabírají v I/O prostoru 3 uzly, jeden pro každý volitelný I/O modul. Může tedy být na sběrnici zapojeno teoreticky až 80 jednotek MEX, tj. odhadem cca 3 800 vstupů/výstupů.

Číslování uzlů

Je standardizované u PLC MPC400 a jednotek MEX400:

0 I/O modul v hlavním PLC na 1.pozici (vpravo)

1 I/O modul v hlavním PLC na 2.pozici (uprostřed)

2 I/O modul v hlavním PLC na 3.pozici (vlevo)

3..15 Nevyužité, rezervované adresy

16..255 Adresy uzlů (zařízení) na lince EXbus - jednotky MEX400 apod.

Pokud se jedná o další členy řady a technologicky příbuzné a kompatibilní typy je číslování specifické a odkazujeme na specializovanou dokumentaci o ovladačích a technické listy výrobků.

Pozn.: Jednotky MEX400 mají kapacitu pro umístění až 3 volitelných I/O modulů (stejně jako MPC400) a zabírají na EXbus tolik adres uzlů, kolik je v nich umístěno modulů. Jednotkám MEX se vždy nastavuje tzv. bázová adresa - na této adrese se nachází I/O modul na 1.pozici vpravo. Další I/O moduly, pokud jsou v jednotce instalovány, pak následují vzestupně. Je vhodné adresovat jednotky MEX400 vždy v odstupu 3 adres, i když nemají plný počet I/O modulů. Zvýší to přehlednost a usnadní pozdější rozšiřování..

Vstupy/výstupy ve zdrojovém textu SIMPLE4

Používání referencí na I/O je asi hlavní a nejzásadnější změnou nové řady 400 proti stávajícím konvencím u PLC MICROPEL. Data k jednotlivým I/O uzlům (zařízení, resp. I/O modul = jeden uzel) jsou uspořádána do struktur a těch je za sebou uloženo v poli 256. Každý typ zařízení má svoji datovou strukturu se specifickým umístěním svých vstupů/výstupů a dalších pomocných nebo konfiguračních registrů. Datová pole ke všem typům I/O modulů jsou již předefinována, pro přístup ke konkrétnímu I/O modulu je jen třeba zvolit vždy odpovídající datovou strukturu. Datové struktury k jednotlivým modulům jsou nazvány podle modulů - **ioa** pro modul A, **iof** pro modul F atd... (blíže viz DATOVÉ STRUKTURY).

Ukázka nastavení digitálního výstupu Y5 na modulu A na poslední (2.) pozici základního PLC:

```
ioa[2].y?5 = 1
```

Ukázka použití digitálního výstupu Y0, analogového vstupu I4 a konfiguračního registru ADCMODE na modulu D s adresou 25 na lince EXbus:

```
iod[25].adcmode = 16
iod[25].y?0 = 1
if iod[25].i[4] > 3000 then Alarm()
```

Digitální vstupy/výstupy

Pokud má dané zařízení EXbus nějaké digitální vstupy (nebo univerzální vstupy s alternativní digitální funkcí), pak je v jeho datové struktuře přítomna položka "**.x**".

Pokud má dané zařízení EXbus nějaké digitální výstupy (nebo univerzální výstupy s alternativní digitální funkcí), pak je v jeho datové struktuře přítomna položka "**.y**".

Dle počtu těchto vstupů (výstupů) může být položka **x / y** typu byte, word nebo longword:

1-8 vstupů/výstupů: **byte**, 9-16 vstupů/výstupů: **word**, 17-32 vstupů/výstupů: **longword**

Jednotlivé digitální vstupy/výstupy jsou v položkách x/y řazeny shodně s číslováním fyzických I/O jako bity ve směru od bitu 0 k vyšším bitům a lze k nim přistupovat otazníkovou konvencí (bitový přístup) jazyka SIMPLE4:

```
ioa[0].y?0 = 1          ; výstup Y0, modul A na 1.pozici v PLC
ioa[0].y?7 = 1          ; výstup Y7, modul A na 1.pozici v PLC
MRAZ = iog[1].x?12      ; vstup X12, modul G na 2.pozici v PLC
```

Takovéto sloučení bitů do jediného bytu/wordu/longwordu má svoje výhody, je např. možné naráz přečíst nebo nastavit celou skupinu vstupů/výstupů:

```
var word  DI_modulG     ; definování proměnných
var byte  DO_modulA
DI_modulG = iog[1].x    ; operace se vstupy/výstupy
ioa[0].y = DO_modulA
```

Pro potřeby programu je z hlediska celkové přehlednosti a budoucí údržby programu maximálně užitečné pojmenovat vstupy/výstupy symbolickými názvy:

```
iog[1].x?12 #  DI_MRAZOVKA
ioa[0].y?1  #  DO_CERPADLO
```

Pozn.: Při důsledném využívání symbolického pojmenování vstupů/výstupů v programech se stane i přenos stávajících hotových aplikací ze starší řady MPC300 na MPC400 snadnou záležitostí. Většinou stačí jen změnit úvodní symbolické definice vstupů/výstupů.

Analogové vstupy/výstupy

Pokud má dané zařízení EXbus nějaké analogové (nebo univerzální) vstupy, pak je v jeho datové struktuře přítomno pole položek "**.i[n]**", kde n je max.počet analogových vstupů na zařízení.

Pokud má dané zařízení EXbus nějaké analogové (nebo univerzální) výstupy, pak je v jeho datové struktuře přítomno pole položek "**.o[n]**", kde n je max.počet analogových výstupů na zařízení.

Standardní analogové vstupy/výstupy mají rozměr 16 bitů a položky jsou typu **word**.

Typický způsob přístupu k nim:

```
Teplota = iod[0].i[1]   ; ANL vstup I1 modulu D na 1.pozici MPC400
iof[1].o[4] = FMvent    ; ANL výstup O4 modulu F na 2.pozici MPC400
```

A ještě příklad pojmenování vstupů/výstupů symbolickými názvy:

```
iod[0].i[1] #  AI_TEPLOTA
iof[1].o[0] #  AO_FMVENT
```

Ostatní registry zařízení

Různá zařízení a moduly I/O se složitější funkcí mohou mít kromě I/O registrů i další registry, např. moduly pro měření teplot mají registr ADCMODE pro nastavení režimu měření apod.

Přístup k nim je podobný jako k analogovým vstupům/výstupům, je-li takový registr v zařízení jen jeden, pak není umístěn v poli ale samostatně. Např. konfigurační registr ADCMODE:

```
iod[0].adcmode = 24     ; měření teploty, převod na 0.1K, 2-vodičově
```

I/O moduly pro řadu 400

Jsou stejné pro MPC400 i MEX400. Jejich označení, funkce a možnosti se shodují s moduly pro řadu MPC300.

Důležitá odlišnost:

V modulech pro MPC400 již nejsou kalibrační registry, protože MPC400 ve spojení se SIMPLE4 disponují prostředky pro provádění dostatečně přesných přepočtů.

modul A

Fyzické I/O :	**8x DI, 8x DO**	
Data DI :	**byte x**	(x?0 .... x?7)
Data DO :	**byte y**	(y?0 .... y?7)
Data AI :	**- - -**	
Data AO :	**- - -**	
Ostatní :	**- - -**	

ioa[*n*].x	8x DI - digitální vstupy (X0..X7)	bitově: ioa[*n*].x?0 .... ioa[*n*].x?7
ioa[*n*].y	8x DO - digitální výstupy (Y0..Y7)	bitově: ioa[*n*].y?0 .... ioa[*n*].y?7

kde n značí číslo (adresu) uzlu, resp. I/O modulu

- **modul B**
- **modul C**

Fyzické I/O :	**8x AI, 8x DO**
Data DI :	**- - -**
Data DO :	**byte y** (y?0 .... y?7)
Data AI :	**word[8] i**
Data AO :	**- - -**
Ostatní :	**byte adcmode**

Pro modul B:

iob[*n*].y 8x DO - digitální výstupy (Y0..Y7) bitově: iob[*n*].y?0 .... iob[*n*].y?7

iob[*n*].i[0..7] 8x AI - analogové vstupy (I0..I7)

iob[*n*].adcmode konfigurační registr ADCMODE pro AI

Pro modul C:

ioc[*n*].y 8x DO - digitální výstupy (Y0..Y7) bitově: ioc[*n*].y?0 .... ioc[*n*].y?7

ioc[*n*].i[0..7] 8x AI - analogové vstupy (I0..I7)

ioc[*n*].adcmode konfigurační registr ADCMODE pro AI

kde n značí číslo (adresu) uzlu, resp. I/O modulu

Registr ADCMODE má strukturu:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
-	-	-	ADC_INPUTS			ADC_FAST	

ADC_FAST – volí rychlost převodu a přesnost zpracování vstupních signálů

- 0 – pomalu, 50 vzorků, kompatibilní se starší verzí, kde ADC_FAST = 0
- 2 – rychle, 10 vzorků, kompatibilní se starší verzí, kde ADC_FAST = 1
- 1 – extra rychle, 5 vzorků
- 3 – jednotlivě, jeden vzorek

Ve všech režimech probíhá vzorkování v rastru 2ms. Režimy 0 a 2 jsou kompatiblní se staršími verzemi FW pro analogové vstupy a zajišťují potlačení rušení 50Hz v napájecí síti. Režimy 1 a 3 s menším počtem vzorků urychlují zpracování vstupních signálů za horší odolnosti proti rušení.

ADC_INPUTS – volí počet zpracovávaných vstupních signálů počínaje vstupem I0

- 0 – všech 8 vstupů tj. I0, I1, I2, I3, I4, I5, I6 a I7
- 1 – vstup I0
- 2 – vstupy I0 a I1
- 3 – vstupy I0, I1 a I2
- 4 – vstupy I0, I1, I2 a I3
- 5 – vstupy I0, I1, I2, I3 a I4
- 6 – vstupy I0, I1, I2, I3, I4 a I5
- 7 – vstupy I0, I1, I2, I3, I4, I5 a I6

Počet vstupů ADC_INPUTS = 0 je nastavení kompatibilní se staršími verzemi FW pro analogové vstupy, kdy se vždy zpracovávalo všech 8 vstupů modulu.

Všechna nová nastavení směřují k možnosti urychlit zpracování vstupních signálů pro potřeby realizace rychlých analogových vstupů. Perioda zpracování vstupního signálu se stanoví:

t = počet_vstupů x počet_vzorků x 2 [ms]

Příkad:

- kompatibilní nastavení pomalu pro celý modul tj. 8 vstupů
  t = 8 x 50 x 2 = 800 [ms]
- kompatibilní nastavení rychle pro celý modul tj. 8 vstupů
  t = 8 x 10 x 2 = 160 [ms]
- nastavení extra rychle pro celý modul tj. 8 vstupů
  t = 8 x 5 x 2 = 80 [ms]
- nastavení jednotlivě pro celý modul tj. 8 vstupů
  t = 8 x 1 x 2 = 16 [ms]
- kompatibilní nastavení pomalu pro 2 vstupy tj. I0 a I1
  t = 2 x 50 x 2 = 200 [ms]
- kompatibilní nastavení rychle pro 2 vstupy tj. I0 a I1
  t = 2 x 10 x 2 = 40 [ms]
- nastavení extra rychle pro 2 vstupy tj. I0 a I1
  t = 2 x 5 x 2 = 20 [ms]
- nastavení jednotlivě pro 2 vstupy tj. I0 a I1
  t = 2 x 1 x 2 = 4 [ms]

  Málokdy je potřeba využít na modulu všech vstupních signálů a proto můžeme konfigurací zpracování zvýšit jeho rychlost pro každý vstup redukcí počtu vstupů, přičemž přesnost a odolnost převodu vůči rušení může zůstat zachována.

- **modul D**
- **modul E**

Fyzické I/O :	**6x AI, 8x DO**
Data DI :	**- - -**
Data DO :	**byte y** (y?0 .... y?7)
Data AI :	**word[6] i**
Data AO :	**- - -**
Ostatní :	**byte adcmode**

Pro modul D:

iod[*n*].y 8x DO - digitální výstupy (Y0..Y7) bitově: iod[*n*].y?0 .... iod[*n*].y?7

iod[*n*].i[0..5] 6x AI - analogové vstupy (I0..I5)

iod[*n*].adcmode konfigurační registr ADCMODE pro AI

Pro modul E:

ioe[*n*].y 8x DO - digitální výstupy (Y0..Y7) bitově: ioe[*n*].y?0 .... ioe[*n*].y?7

ioe[*n*].i[0..5] 6x AI - analogové vstupy (I0..I5)

ioe[*n*].adcmode konfigurační registr ADCMODE pro AI

kde n značí číslo (adresu) uzlu, resp. I/O modulu

Registr ADCMODE má strukturu:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
-	-	-	ADC_2W	ADC_Temp	-	ADC_Type	

ADC_Temp 0 = měření odporu, 1 = převod na teplotu v 0.1K (Kelvin)

ADC_2W 0 = 3-vodičové měření, 1 = 2-vodičové měření

ADC_Type typ převodu na teplotu, má význam jen u modulu E, viz tab:

ADC_Type	Typ přepočtu
0	Pt100/Pt1000
1	Ni5000
2	Ni6180
3	Pt100/Pt1000

modul F

Fyzické I/O :	**8x DI, 6x AO**
Data DI :	**byte x** (x?0 .... x?7)
Data DO :	- - -
Data AI :	- - -
Data AO :	**word[6] o**
Ostatní :	- - -

iof[*n*].x 8x DI - digitální vstupy (X0..X7) bitově: iof[*n*].x?0 .... iof[*n*].x?7

iof[*n*].o[0..5] 5x AO - analogové výstupy (O0..O5)

kde n značí číslo (adresu) uzlu, resp. I/O modulu

modul G

Fyzické I/O :	**16x DI**
Data DI :	**word x** (x?0 .... x?15)
Data DO :	- - -
Data AI :	- - -
Data AO :	- - -
Ostatní :	- - -

iog[*n*].x 16x DI - digitální vstupy (X0..X15) bitově: iog[*n*].x?0 .... iog[*n*].x?15

kde n značí číslo (adresu) uzlu, resp. I/O modulu

- **modul H**
- **modul I**

Fyzické I/O :	**8x DI, 8x AI**
Data DI :	**byte x** (x?0 .... x?7)
Data DO :	- - -
Data AI :	**word[8] i**
Data AO :	- - -
Ostatní :	**byte adcmode**

Pro modul H:

ioh[*n*].x 8x DI - digitální vstupy (X0..X7) bitově: ioh[*n*].x?0 .... ioh[*n*].x?7

ioh[*n*].i[0..7] 8x AI - analogové vstupy (I0..I7)

ioh[*n*].adcmode konfigurační registr ADCMODE pro AI

Pro modul I:

ioi[*n*].x 8x DI - digitální vstupy (X0..X7) bitově: ioi[*n*].x?0 .... ioi[*n*].x?7

ioi[*n*].i[0..7] 8x AI - analogové vstupy (I0..I7)

ioi[*n*].adcmode konfigurační registr ADCMODE pro AI

kde n značí číslo (adresu) uzlu, resp. I/O modulu

Registr ADCMODE má strukturu:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
-	-	-	ADC_INPUTS			ADC_FAST	

ADC_FAST – volí rychlost převodu a přesnost zpracování vstupních signálů

ADC_INPUTS – volí počet zpracovávaných vstupních signálů počínaje vstupem I0

Detailní popis je uveden v kapitole **modul B, modul C**.

- **modul J**
- **modul K**

Fyzické I/O :	**8x DI, 6x AI**
Data DI :	**byte x** (x?0 .... x?7)
Data DO :	- - -
Data AI :	**word[6] i**
Data AO :	- - -
Ostatní :	**byte adcmode**

Pro modul J:

ioj[*n*].x 8x DI - digitální vstupy (X0..X7) bitově: ioj[*n*].x?0 .... ioj[*n*].x?7

ioj[*n*].i[0..5] 6x AI - analogové vstupy (I0..I5)

ioj[*n*].adcmode konfigurační registr ADCMODE pro AI

Pro modul K:

iok[*n*].x 8x DI - digitální vstupy (X0..X7) bitově: iok[*n*].x?0 .... iok[*n*].x?7

iok[*n*].i[0..5] 6x AI - analogové vstupy (I0..I5)

iok[*n*].adcmode konfigurační registr ADCMODE pro AI

kde n značí číslo (adresu) uzlu, resp. I/O modulu

Registr ADCMODE má strukturu:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
-	-	-	ADC_2W	ADC_Temp	-	ADC_Type	

ADC_Temp 0 = měření odporu, 1 = převod na teplotu v 0.1K (Kelvin)

ADC_2W 0 = 3-vodičové měření, 1 = 2-vodičové měření

ADC_Type typ převodu na teplotu, má význam jen u modulu K, viz tab:

ADC_Type	Typ přepočtu
0	Pt100/Pt1000
1	Ni5000
2	Ni6180
3	Pt100/Pt1000

modul L

Fyzické I/O :	**8x FDI, 8x DO**
Data DI :	**byte fx** (fx?0 .... fx?7)
Data DO :	**byte y** (y?0 .... y?7)
Data AI :	- - -
Data AO :	- - -
Ostatní :	**word[8] cnt** **word[4] irc**

iol[*n*].fx	8x FDI - rychlé dig. vstupy (X0..X7)	bitově: iol[*n*].fx?0 ....iol[*n*].fx?7
iol[*n*].y	8x DO - digitální výstupy (Y0..Y7)	bitově: iol[*n*].y?0 .... iol[*n*].y?7
iol[*n*].cnt[0..7]	8x čítače příslušné k rychlým vstupům FDI (X0..X7)	
iol[*n*].irc[0..3]	4x obousměrné čítače pro inkrementální čidla (vždy pro dvojice vstupů FDI: X0/X1, X2/X3, X4/X5, X6/X7)	

kde n značí číslo (adresu) uzlu, resp. I/O modulu

Pozn.: Modul L je specifický pro řadu 400, nelze jej použít pro MPC300.

modul M

Fyzické I/O :	**8x FDI, 8x DI**
Data DI :	**byte fx** (fx?0 .... fx?7) **byte x** (x?0 .... x?7)
Data DO :	- - -
Data AI :	- - -
Data AO :	- - -
Ostatní :	**word[8] cnt** **word[4] irc**

iom[*n*].fx	8x FDI - rychlé dig. vstupy (X0..X7)	bitově: iom[*n*].fx?0 ....iol[*n*].fx?7
iom[*n*].x	8x DI - digitální vstupy (X8..X15)	bitově: iom[*n*].x?0 .... iol[*n*].x?7
iom[*n*].cnt[0..7]	8x čítače příslušné k rychlým vstupům FDI (X0..X7)	
iom[*n*].irc[0..3]	4x obousměrné čítače pro inkrementální čidla	

(vždy pro dvojice vstupů FDI: X0/X1, X2/X3, X4/X5, X6/X7)

kde n značí číslo (adresu) uzlu, resp. I/O modulu

Pozn.: Modul M je specifický pro řadu 400, nelze jej použít pro MPC300.

Bezpečnostní funkce

S programováním aplikací a nasazením rozšiřovacích jednotek MEX400 souvisí i zavedené a podporované bezpečnostní funkce, které spočívají v řešení havarijních a inicializačních stavů souvisejících s komunikací po lince Exbus.

- **zapnutí napájení celého systému** – znamená, že všechny automaty i rozšiřující jednotky budou po nějakou dobu ustanovovat komunikaci a její parametry po lince EXbus. Z toho plyne, že po tuto dobu nemá centrální PLC kontrolu nad vstupy a výstupy podřízených jednotek. Proto má každý výstup na rozšiřujících jednotkách definován stav (hodnotu) po zapnutí napájecího napětí. Tato hodnota je továrně nastavována na 0 tj. všechny digitální výstupy jsou vypnuty a všechny analogové výstupy jsou staženy na minimum.
- **reset komunikační linky** – k němuž dochází v průběhu zatahování nového uživatelského kódu do centrálního PLC tj. v okamžiku, kdy PLC sběrnici neobsluhuje a ani neběží žádný aplikační program. Hodnoty, na které se nastavují jednotlivé výstupy v případě resetu linky jsou továrně nastaveny na 0 a funkce inicializace výstupů při resetu linky je aktivována.
- **ztráta komunikace** – k níž může docházet vlivem poškozeného, zarušeného nebo impedančně nepřizpůsobeného komunikačního vedení ale i z jiných příčin, je chápána jako specifický havarijní stav. Od tohoto stavu je možné opět nastavit hodnoty výstupů na předem definované. Krom toho je u této funkce možné nastavit i čas po jehož uplynutí se havarijní funkce aktivuje. To je proto, že stav výpadku komunkace je detekován podřízenou jednotkou podle četnosti její obsluhy na lince EXbus. Pokud bude na této lince velké množství podřízených jednotek, pak cyklus obsluhy všech IO má delší periodu než v případě, že na lince bude např. pouze jeden rozšiřující modul. Továrně je tato funkce povolena, výstupy se nastavují do 0 a čas detekce výpadku je nastaven na 1s.

Pokud nastavení hodnot z nějakého důvodu nevyhovuje, může ho programátor změnit odděleně u každé stanice MEX400 a každého výstupu pomocí programu dataserver.

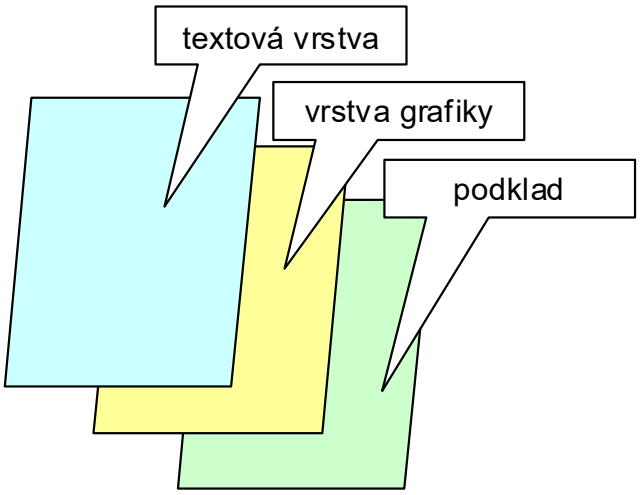
Bezpečnostní funkce ze strany centrálního PLC je řešena pouze pomocí pomocí bitového pole iolist (viz. Doplňkové funkce I/O subsystému). Pole udává seznam aktuálně připojených uzlů, takže je možné tento seznam porovnávat vůči topologii aplikace a vyhodnotit tak možné poruchy, jako jsou výpadky připojení, odmlčení stanice atp.

3. ZOBRAZOVACÍ SYSTÉM

PLC řady 400 přicházejí s novým unikátním zobrazovacím systémem MGD (MICROPEL Graphic Display), který poskytuje rozsáhlé nové možnosti pro tvorbu grafiky a zároveň zachovává úplnou kompatibilitu pro stávající aplikace vytvořené pro textové displeje.

Vrstvy

Výsledný obraz na displeji je v reálném čase neustále skládán ze tří virtuálních vrstev, umístěných za sebou. Všechny vrstvy mají stejný rozměr, jako je rozměr displeje. MGD pracuje (v závislosti na typu vrstvy) buď s 256 nebo 65536 barvami. Jeden kód barvy je vždy vyhrazen pro barvu průhlednou - pokud je v některé vrstvě použita, pak se pixely vyplněné touto barvou stávají průhledné a je zde vidět vrstva která je umístěná níže (situaci znázorňuje nákres). Pokud tedy např. v horní vrstvě nebude nikde použita průhledná barva, nebudou další vrstvy viditelné (jako by neexistovaly). A pokud bude naopak celá horní vrstva vyplněna průhlednou barvou, potom "zmizí".



Vrstva TEXT

Typ :	**TEXTOVÁ VRSTVA**
Umístění :	**úplně nahoře**
Rozlišení :	**až 16 x 40 znaků**
Barvy :	**8-bitové (paleta 256 barev)**
Po resetu :	smazaná (mezery), barva: neprůhledná, pozadí-černá, písmo-bílá

Umísťují se do ní znaky, nejmenší polohové rozlišení je 1 znak. Znaky mají pevný rozměr 8x15 pixelů (na výšku), rastr pro umísťování znaků je též pevný.

Celá tato vrstva funguje stejným způsobem jako znaková textová obrazovka používaná u automatů MICROPEL K1, K10, MPC300. Má identické ovládání přes proměnnou POSITION a sadu funkcí DISPLAY. Výhodou navíc je větší počet znaků, možnost volby barev a znaková sada s úplnou českou diakritikou.

Hlavní předností textové obrazovky je její extrémně jednoduché ovládání.

Pozn.: Po restartu PLC jsou barvy textové obrazovky neprůhledné, automat se tedy chová jako by měl pouze textovou obrazovku. Pokud totiž text vymažeme pomocí znaku "mezera", bude tento vykreslen barvou podkladu tj. černě a vrstva grafiky a podkladová vrstva tak zůstanou překryty. Aniž by tedy uživatel cokoli nastavoval, může spustit např. na MPC405 program, který byl původně určený pro MPC300 nebo K1/K10, případně MT201. Výjimkou jsou pouze uživatelsky definované znaky, které mají proti předchozím typům PLC MICROPEL jiné (větší) počty pixelů vyplývající z jiné velikosti systémového fontu a u některých výrobků nejsou vůbec podporovány.

Pro vyšší přehlednost mají všechny funkce SIMPLE4 pro operace s textovou vrstvou názvy s předponou **Display**...

Vrstva GRAPHIC

Typ :	**GRAFICKÁ VRSTVA PRO KRESLENÍ**
Umístění :	**uprostřed**
Rozlišení :	**až 800 x 480 pixelů**
Barvy :	**8-bitové (paleta 256 barev)**
Po resetu :	smazaná - vyplněná průhlednou barvou

Hlavní vrstva pro kreslení grafických prvků. Pro uživatelské programy v SIMPLE4 je k dispozici široký výběr funkcí pro kreslení základních geometrických tvarů, různě velkých obrázků (bitmap) i s možností animace. I do této vrstvy lze tisknout texty, dokonce různě velkými fonty i s různým otočením a s jemně určenou polohou. Nejmenší rozlišení pro kreslení a umísťování objektů je 1 pixel. Protože všechny objekty jsou vytvářeny voláním vestavěných funkcí, je možné libovolně naprogramovat i chování grafiky (změny a pohyb objektů po obrazovce v reálném čase apod.). Bitmapy pro tuto vrstvu mohou být libovolně velké od drobných ikon až po obrázky (max. do rozměru obrazovky), musí být kódované do 256 barev.

Pro vyšší přehlednost mají všechny funkce SIMPLE4 pro operace s grafickou vrstvou názvy s předponou **Gdi**...

Aby grafická vrstva byla na displeji viditelná, musí být horní vrstva TEXT vyplněna průhlednou barvou (buď celá, nebo jen tam kde je to třeba).

TIP:

Pokud bude na textové obrazovce nastavena barva pozadí průhledná a barva písma nějaká kontrastní (černá, bílá, žlutá - podle konkrétní situace), pak mohou fungovat obě vrstvy současně - na obrazovce bude vidět grafická vrstva s nakreslenými objekty, obrázky či fotografií a přes ni bude výrazně (např. bíle) zobrazováno písmo z horní textové obrazovky.

Vrstva BACKGROUND

Typ :	**GRAFICKÁ VRSTVA S PODKLADEM**
Umístění :	**úplně dole**
Rozlišení :	**až 800 x 480 pixelů**
Barvy :	**16-bitové (65536 barev), nebo 8-bitové (paleta 256 barev)**
Po resetu :	smazaná - nastavená na plnou černou barvu

Lze jí přiřadit buď jednu pevnou barvu, nebo lze do ní umístit podkladový obrázek (bitmapu). Bitmapa může mít buď 256 nebo 65536 barev. Tato vrstva neumožňuje žádné kreslení, ale je např. ideální pro umístění fotografie, protože jako jediná ze 3 vrstev systému MGD zobrazuje 65536 barev. Přiřazení obrázku k podkladové vrstvě se provádí funkcemi SIMPLE4, lze tedy podkladové obrázky i za běhu programu měnit.

Aby podkladová vrstva byla na displeji viditelná, musí být v daném místě obě vyšší vrstvy (TEXT i GRAPHIC) vyplněny průhlednou barvou. **!POZOR!** Pokud je podkladem obrázek, musí mít přesně rozměr, který odpovídá rozlišení displeje.

Protože se podkladový obrázek vykresluje přímo vnitřními funkcemi firmware, nezpomalí jeho vykreslení běh hlavní programové smyčky na rozdíl od kreslení obrázku do grafické vrstvy.

Pozn.: protože podkladová vrstva je poslední v řadě, logicky už nemůže být průhledná. Pokud tu tedy bude nastavena průhledná barva, bude interpretována maximální zelenou (16-bitový kód barvy 0x07E0).

TIP: Zajímavým řešením pro velmi efektní a přitom programátorsky nenáročný styl zobrazení může být použití pouze horní a spodní vrstvy (TEXT + BACKGROUND):

- do vrstvy BACKGROUND vložit podkladovou fotografii v 65536 barvách

- prostřední vrstvu GRAPHIC nepoužívat, jen kompletně vyplnit průhlednou barvou

- v horní vrstvě TEXT nastavit průhlednou barvu pozadí a výraznou barvu textu

Tímto způsobem lze pak v textové vrstvě jednoduše provozovat klasické zobrazování a textové řádkové menu stejně jako např. u PLC K1/K10/MPC300, přičemž texty budou zobrazovány přes podkladový obrázek.

Barvy

MGD pracuje s dvojím kódováním barev: 8-bitovým (256 barev) a 16-bitovým (65536 barev). Některé vrstvy mohou podporovat obě kódování, některé jen jedno (viz stať Vrstvy).

To, o jaký typ barvy jde, je určeno datovým typem příslušné proměnné - pro 8-bitovou barvu se užívá datový typ byte a pro 16-bitovou typ word.

V obou případech je jedna kódová kombinace vyhrazena pro zadání "průhledné barvy", resp. pro zprůhlednění dané vrstvy.

16-bitové barvy

Celkem k dispozici: 65535 barev + 1 průhledná. Jedná se o základní barevnou soustavu pro systém MGD na PLC MPC405. Kódování barev je **RGB-5:6:5** (podle počtu bitů vyhrazených pro jednotlivé složky R, G, B). 16-ti bitový kód barvy je dán výrazem:

COLOR = R*2048 + G*32 + B, strukturu kódu ukazuje tabulka.

bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
význam:	R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0
složka:	červená					zelená						modrá				

Kde složky RGB mohou nabývat hodnot: R=0...31, G=0...63, B=0...31

Průhledná barva = 0x07E0 (zde definována místo plné zelené: R=0, G=63, B=0)

8-bitové barvy

Celkem k dispozici 255 barev + 1 průhledná. Zde má kódování trochu jiný význam. Kód totiž implicitně neznamená podíl složek RGB jako v případě 16-bitové barvy, ale je to index do předdefinované palety 256-ti barev. Paletou se rozumí přiřazovací tabulka o 256 řádcích, kde každý řádek obsahuje jednu vybranou 16-bitovou barvu (viz 16-bitové barvy).

Průhledná barva = 0xFF (zde tedy zabírá poslední řádek tabulky)

Proč tak složitě a proč se tedy nepoužívají jen 16-bitové barvy?

Protože kreslení a zpracování grafiky v 16-bitových barvách je mnohem náročnější na paměťové prostory - jak datové (potřebné pro běh zobrazování), tak programové (potřebné bitmapy se ukládají do paměti programu FLASH EPROM). A primární úlohou PLC řady 400 je přece jen více řízení a regulace, než grafické zobrazení.

Pro vektorové kreslení grafických objektů, pro texty a pro menší bitmapy je 8-bitová barevná hloubka dostatečná, pro zobrazení fotografií nebo bitmap s velkým množstvím polotónů je tu vrstva BACKGROUND, která zobrazuje plných 65536 barev.

Konstrukce palety 256 barev v systému MGD:

Prvních 216 barev se lineárně skládá ze složek RGB, které mohou nabývat hodnot:

R=0...5, G=0...5, B=0...5

Kód barvy (0...215) je dán výrazem COLOR = R*36 + G*6 + B.

Dále následují barvy uživatelské a barvy pevně zadefinované.

Poslední 3 barvy z palety jsou:

253 - bílá, 254 - černá, 255 - průhledná

Bitmapy

Všechny definované bitmapy (rastrové obrázky) jsou ukládány společně s uživatelským programem do kódové paměti. Pokud je jich mnoho nebo jsou relativně velké, zaberou nezanedbatelnou část programové paměti. Je tedy dobré vědět, kolik místa zhruba potřebují.

druh obrázku	rozměr[pix]	počet barev	potřeba paměti[B]
malá ikona, symbol, piktogram	30 x 30	256	900
obrázek, diagram - na 1/4 displeje	52 x 100	256	5 200
kreslený obr., schema - na celý displej	105 x 200	256	21 000
fotografie - na celý displej	105 x 200	65536	42 000

Pokud tedy aplikační program pro MPC405 zabere např. 200kB programové paměti, pak z celkových 768kB zbývá ještě 568kB, kam lze složit např. 13 fotografií na celý displej v plných barvách, nebo cca 100 obrázků na 1/4 displeje v 256 barvách.

Chod zobrazování

Přenos grafické informace na displej je prováděn dvěma nezávislými procesy:

1. Konstrukce grafiky v paměti

Proces je zcela v režii programátora a tvoří jej sekvence volání grafických a textových funkcí programovacího jazyka SIMPLE4 pro výstup grafiky a textu do vrstev TEXT a GRAPHIC.

Sekvence probíhá vždy v rámci běhu programové smyčky uživatelského programu. Výstup všech těchto funkcí jde do virtuální grafické obrazovky v paměti - tzv. grafické stránky.

2. Výstup grafiky na displej

Tento proces probíhá neustále na pozadí a zajišťuje stálý přenos grafiky z virtuální obrazovky v paměti na fyzický displej PLC. U MPC405 trvá přenos jedné kompletní obrazovky typicky asi 50-60ms. Virtuální grafické obrazovky v paměti jsou dvě - pro ilustraci např. A a B. Do jedné vždy probíhá kreslení grafických objektů uživatelským programem (1.proces) a druhou má k dispozici 2.proces, který provádí výstup na displej. Pokud 1.proces dokončil konstrukci grafiky v obrazovce A (a to je zpravidla na 1 průchod hlavní smyčky), nemá už k ní přístup a systém MGD nastavuje příznakový bit **GDIBUSY**=1. Čeká se až 2.proces dokončí přenos finální obrazovky B složené ze tří vrstev grafiky (viz. výše) na displej. V tomto stavu budou nyní všechny grafické GDI-funkce naprázdno propadávat, aby zbytečně nezdržovaly chod programu, protože stejně nemá smysl pořád znovu kreslit grafiku, není-li ještě předchozí grafická obrazovka přenesena na displej. Pokud 2.proces dokončí výstup obrazovky B na displej dojde k uvolnění kreslení na obrazovku A a nastaví příznak **GDIBUSY**=0. Pokud je příznak **GDI_KEEP**=0, pak se na konci běhu hlavní smyčky obrazovka A kompletně smaže (vyplní průhlednou barvou) a nuluje se příznak **GDIBUSY**=0. Pokud je nastaven příznak **GDI_KEEP**=1, pak se obrazovka A nemaže a zůstává na ní nakreslen původní obraz. Příznak GDI_KEEP tak najde uplatnění např. při konstrukci "vyskakovacích oken editorů", kdy se využívá k zobrazení editované hodnoty pouze malá plocha displeje.

Kreslení na více průchodů hlavní smyčkou

Někdy je potřebné nebo užitečné rozdělit vykreslení grafiky na více průchodů uživatelského programu hlavní smyčkou. Mohou k tomu vést tyto důvody:

- Vytvářená grafika je velmi složitá a volání všech potřebných funkcí v jednom průchodu smyčky by mohlo způsobit velké prodloužení doby průchodu smyčkou. To by např. při sledování a řízení rychlých dějů mohlo působit komplikace, proto může být výhodnější rozložit kreslení grafiky do několika průchodů smyčkou. Informaci o tom, kolik času zabere volání všech grafických funkcí ve smyčce, může podat funkce GdiThreadTime (viz popis grafických funkcí).
- Vytvářená grafika obsahuje množství identických nebo podobných objektů. Zde může být výhodné vytvořit cyklus, kde v každém průchodu smyčky bude vykreslen jeden objekt (či skupina objektů).

V těchto případech je potřebné informovat systém MGD, že obraz v paměti ještě není hotový a že na konci hlavní smyčky ještě nesmí dojít k uzavření grafické stránky na kterou se zrovna kreslí. K tomu slouží bit **GDIHOLD**. Ten je vždy na začátku smyčky systémem vynulován. Má-li se tedy dokončení stránky pozdržet, je nutné v každém takovém průchodu smyčky nastavit **GDIHOLD**=1. Až ve finálním posledním průchodu se **GDIHOLD** ponechá bez změny a tím se stránka na konci smyčky předá k vykreslení.

Aby se správně napočítaly průchody, kdy se grafika kreslí, je ještě nutné sledovat bit **GDIBUSY**. Pokud je totiž **GDIBUSY**=1, do grafické stránky se nekreslí, grafické funkce by propadávaly naprázdno a tedy ani nemá smysl je volat, ani posouvat počítadlo úspěšných průchodů kreslení grafiky (protože k žádnému kreslení stejně nedochází).

Ilustrační příklad:

Vykreslení padesáti stejných objektů (modrý čtverec, bílá linka) do matice, na 50 průchodů:

```
var byte ObjCNT                     ; počítadlo objektů
var pxy ObjPosition                 ; proměnná pro uložení pozice objektu
code pxy ObjSize = (16,16)          ; pevně daný rozměr objektu (16x16 pixelů)
      ;---------------- HLAVNÍ SMYČKA ---------------------
if RESET then begin                 ; Inicializace po resetu
  ObjCNT=0                          ; nulovat čítač objektů
  DisplayBColor(c_none)             ; průhledná barva pozadí pro vrstvu TEXT
  DisplayClear()                    ; vyplnění vrstvy TEXT barvou pozadí
end
ObjPosition.x=(ObjCNT%10)*20+1      ; pozice X (10 objektů s odstupem 20pix)
ObjPosition.y=(ObjCNT/10)*20+4      ; pozice Y (po 10 objektech nový řádek)
GdiFColor(c_blue)                   ; nastavit modrou pro vrstvu GRAPHIC
GdiFillRect(ObjSize,ObjPosition)    ; nakreslit plný čtverec - modře
GdiFColor(c_white)                  ; nastavit bílou pro vrstvu GRAPHIC
GdiRect(ObjSize,ObjPosition)        ; nakreslit prázdný čtverec - bíle
if GDIBUSY then ObjCNT=0            ; je-li odstavené kreslení, čítač=0
else begin
  ObjCNT=ObjCNT+1                   ; posouvat čítač objektů (i jejich pozice)
  if ObjCNT<50 then GDIHOLD=1       ; dokud není poslední, odkládat vykreslení
end
RESET=0                             ; závěrečné nulování bitu RESET
end.                                ; konec hlavní smyčky
```

4. VESTAVĚNÉ FUNKCE

Používají se při programování systémů v jazyce SIMPLE4. Všemi níže uvedenými funkcemi disponují PLC řady 400 a od ní technologicky odvozené typy již v základní výbavě, bez potřeby nějakých dodatečných knihovních modulů. U všech funkcí jsou v prvé řadě uvedeny jejich deklarace, tak, jak je má ve své konfiguraci uložen překladač SIMPLE4.

Pozn.: Veškeré deklarace funkcí, datových struktur, předdefinované konstanty i datová pole jsou souhrnně umístěny v souboru CONFIG400.INC, v adresáři kde je instalován překladač SIMPLE4, resp. vývojové prostředí StudioWin.

Upozornění: Soubor CONFIG400.INC nepřepisujte ani nijak neupravujte, ohrozil by se tím bezchybný chod překladače a v některých případech i bezchybný běh programu.

Časovací funkce

GetTickCount

Vrací hodnotu typu longword, která představuje počet uplynulých milisekund od startu systému. Pokud počet ms přesáhne hodnotu 4294967295 tj. přibližně 49 dní, dojde k přetečení a odpočet začne znovu od 0.

Deklarace :	**function longword GetTickCount ( )**
Parametr :	- - -
Výstup :	aktuální počet milisekund od startu systému

Funkce se výborně hodí k odměřování takřka libovolného časového úseku a současně můžeme takových úseků odměřovat velmi mnoho. Pro každý z těchto odměřovaných úseků si deklarujeme proměnnou typu longword do níž v okamžiku startu odměřování uložíme počáteční hodnotu pomocí funkce GetTickCount(). Pokud máme hodnotu uloženou, odměříme požadovaný interval (např. 20000ms) pomocí podmíněného příkazu takto:

```
if ((GetTickCout() - start) >= 20000) then ……  ; interval odměřen
```

Systémové funkce

SYS_GetPLCInfo

Vloží do proměnné info (struktura typu _sysinfo_type) základní údaje o nastavení, typu a hw konfiguraci automatu.

Deklarace :	**subroutine SYS_GetPLCinfo (var _sysinfo_type info)**
Parametr :	**info** odkaz na typ proměnné _sysinfo_type, kam bude uložen popis k PLC
Výstup :	- - -

Datová struktura _sysinfo_type je již předdefinována a má tvar:

```
type struct
  longword  SN,              ; sériové číslo
  word      FW,              ; verze FW na 3 des.místa, např. 5004 = FW5.004
  word      PLC_addr,        ; aktuálně nastavená adresa v síti PESnet
  dstring   PLC_type,        ; textové označení typu PLC
  byte      HW_core,         ; kód jádra PLC (rozlišení CPU, rozsahy pamětí)
  byte      HW_id,           ; kód HWID, užívá se pro určení verze hardwaru
  word      S4_instrset,     ; podporovaná instrukční sada Simple4
  longword[27] RES_lw        ; rezerva, doplněno na celkem 40x longword
end   _sysinfo_type
```

Z definice je význam jednotlivých položek zřejmý. Aktuální hodnoty předávané u řady MPC400 v systémových položkách jsou:

```
HW_core = 0x31,   HW_id = 160,   S4_instrset = 200
```

Textové označení typu PLC je bez úvodních písmen MPC a obsahuje i konfiguraci IO:

```
PLC_type = "405A" (MPC405-A)
PLC_type = "405BJL" (MPC405-BJL)
```

Druhá varianta systémové funkce je doplněna o parametr drvid, který představuje identifikační číslo ovladače, jehož parametry od firmwaru požadujeme. Parametry ovladače jsou uloženy v poli bajtů, které začíná na první pozici pole rezervovaných položek RES_lw.

Deklarace :	**subroutine SYS_GetPLCinfo (var _sysinfo_type info, word drvid)**
Parametr 1:	**info** odkaz na typ proměnné _sysinfo_type, kam bude uložen popis k PLC
2:	**drwid** číslo ovladače jehož parametry požadujeme
Výstup :	- - -

Pro zjištění a následné vyhodnocení parametrů ovladače, je vhodné použít následující programovou konstrukci:

```
var _sysinfo_type sys
var byte[108] driver mapon(sys.RES_lw)
```

Tato konstrukce mapuje pole driver na reservované položky struktury sys (_sys_info_type). Pole byte driver bude po zavolání podprogramu SYS_GetPlcInfo obsahovat informace o driveru jehož typ byl při volání specifikován parametrem drvid. Význam obsahu pole driver ukazuje tabulka.

Index	Význam
0	hodnota 0 – driver není instalován, 1 driver je instalován
1	počet platných parametrů ovladače
2..107	parametry ovladače

K dispozici jsou předdefinovaná Id pro nečastěji používané ovladače jejichž symbolické názvy shrnuje tabulka.

Symbol	Význam
_c_pesnet	popis ovladače pesnet pro varianty bez a s automatickým přiřazením rychlosti a adresy
_cpesnet_lite	popis ovladače pesnet-lite parametr rychlost a adresa
_c_exbus_master	popis ovladače exbus master, komunikační rychlost, dělení prostoru uzlů exbus
_c_exbus_slave	popis ovladače exbus slave, bázová adresa, komunikační rychlost
_c_modbus0	popis ovladače modbus na lince 0
_c_modbus1	popis ovladače modbus na lince 1
_c_ethernet	popis ovladače pro přístup na sběrnici ethernet, ip adresa, maska atd.
_c_dma	popis ovladače pro přímý přístup do paměti automatů na lince PESNET
_c_gprs	popis ovladače pro přístup do sítě GSM pomocí SMS a komunikace přes DMA
_c_iobus	popis ovladače pro realizaci uživatelské komunikace master-slave na lince EXBUS
_c_user_eeprom	popis ovladače pro přístup do uživatelské paměti typu EEPROM

Detailní význam jednotlivých byte popisu ovladačů je uveden v textu, věnovaném jednotlivým typům ovladačů. Zde uvedeme pouze ty nejdůležitější a z praktického programátorského hlediska užitečné.

_c_pesnet	
Index	Význam
2	kód aktuální rychlosti komunikace
3	adresa zařízení na lince
4	bit 2 – režim automatické adresy, bit 0 – režim automatické komunikační rychlosti

_c_exbus_master	
Index	Význam
2	kód rychlosti komunikace
3-4	počet vnitřních IO uzlů zařízení
5-6	celkový maximální počet obsluhovaných IO uzlů ovladačem

_c_exbus_slave	
Index	Význam
2	kód rychlosti komunikace
3-4	bázová adresa zařízení

Pro dekódování hodnoty komunikační rychlosti z kódu rychlosti komunikace předávané v popisu ovladače, jsou předdefinovány symboly.

Symbol	Význam
_c_460800bd	rychlost 460.8kBd
_c_230400bd	rychlost 230.4kBd
_c_115200bd	rychlost 115.2kBd
_c_57600bd	rychlost 57.6kBd
_c_38400bd	rychlost 38.4kBd
_c_19200bd	rychlost 19.2kBd
_c_9600bd	rychlost 9.6kBd

SYS_ExbusReset

Funkce provede „teplý“ start komunikační linky EXBUS. Nejprve smaže veškeré tabulky popisující spojení s periferiemi na lince a spustí vyhledávací proces periferií od začátku tak, jak probíhá po restartu automatu.

Deklarace :	**subroutine SYS_ExbusReset ( )**
Parametr :	- - -
Výstup :	- - -

Pozn.: funkce není pro běžné použití třeba, může být vhodná jen ve speciálních případech.

SYS_PLCReset

Funkce způsobí úplný restart automatu. V okamžiku spuštění funkce je uživatelská smyčka programu okamžitě ukončena a bezprostředně na to je automat restartován.

Deklarace :	**subroutine SYS_PLCReset ( )**
Parametr :	- - -
Výstup :	- - -

Pozn.: funkce není pro běžné použití třeba, může být vhodná jen ve speciálních případech.

Funkce pro převod hodnoty odporu na teplotu

Čtveřice funkcí pro převod hodnoty odporu na teplotu, využitelné ve speciálních případech. Analogové vstupy na I/O modulech pro měření odporových teplotních čidel lze nastavit tak, aby

rovnou převáděly měřený odpor na teplotu a dávaly jako výstupní hodnotu přímo teplotu. Existují však případy, kdy je vhodné nastavit analogové vstupy tak, aby poskytovaly hodnotu měřeného odporu v ohmech a pro finální převod na teplotu se použijí uvedené funkce:

- Na vstupy jednoho I/O modulu typu D, E, J, K jsou připojena jednak odporová teplotní čidla, jednak jiné senzory nebo členy u nichž je třeba zjistit buď přímo odpor nebo použít jiný převod odpor/hodnota. Modul se nastaví do režimu měření odporu, pro teplotní čidla se poté použije jedna z uvedených převodních funkcí a ostatní vstupy se vyhodnotí jiným způsobem.
- Na vstupy jednoho I/O modulu typu E, K jsou připojena odporová teplotní čidla s různým průběhem, např. Pt1000, Ni1000/5000ppm, Ni1000/6180ppm. Modul se nastaví do režimu měření odporu, pro příslušná teplotní čidla se pak použije příslušný typ převodní funkce.
- U měřených teplotních čidel se má provádět přesná kompenzace přívodních vodičů a základní odchylky odporu použitého čidla. Modul se nastaví do režimu měření odporu, poté se provedou příslušné výpočty korekcí odporu (součtová korekce na kompenzaci odporu vodičů a násobná korekce kompenzující odchylku odporu teplotního čidla, typicky definovanou při 0.0°C) a po nich se na závěr zařadí příslušné funkce pro převod vypočteného odporu na teplotu.

Výstupní hodnota udávající teplotu je vždy v 0.1K (desetiny Kelvina). Referenční hodnotou pro převod na stupně Celsia je 2732 (273.2K, teplota 0°C v Kelvinech, zaokrouhleno z 273.15).

Příklady přepočtu

výstupní hodnota 2732 = 273.2K (nebo též 0.0°C)

výstupní hodnota 2657 = 265.7K (nebo též -7.5°C)

výstupní hodnota 2952 = 295.2K (nebo též +22.0°C)

Funkce převádějí odpor na teplotu podle charakteristiky čidla bez ohledu na jeho pracovní rozsah teplot. Limitní meze vstupního parametru velikosti odporu je tedy třeba hlídat externě s přihlédnutím k možnostem konkrétního čidla.

R2Pt100

Funkce pro převod hodnoty odporu čidla Pt100 na teplotu.

Deklarace :	**function word R2Pt100 (word odpor)**
Parametr 1 :	**odpor** hodnota odporu pro konverzi [x 0.01 ohm] - např. 13852 = 138.52Ω
Výstup :	**teplota_Kelvin** hodnota teploty v Kelvinech [x 0.1 K] - např. 2932 = 293.2K (+20.0°C)

Odpor je v 0.01 ohm, tedy hodnota 10000 = 100.00 ohm. Výstupní hodnota je v 0.1K (desetiny Kelvina).

R2Pt1000

Funkce pro převod hodnoty odporu čidla Pt1000 na teplotu.

Deklarace :	**function word R2Pt1000 (word odpor)**
Parametr 1 :	**odpor** hodnota odporu pro konverzi [x 0.1 ohm] - např. 12255 = 1225.5Ω
Výstup :	**teplota_Kelvin** hodnota teploty v Kelvinech [x 0.1 K] - např. 2932 = 293.2K (+20.0°C)

Odpor je v 0.1 ohm, tedy hodnota 10000 = 1000.0 ohm. Výstupní hodnota je v 0.1K (desetiny Kelvina).

R2Ni1k5000

Funkce pro převod hodnoty odporu čidla Ni1000/5000ppm na teplotu.

Deklarace :	**function word R2Ni1k5000 (word odpor)**
Parametr 1 :	**odpor** hodnota odporu pro konverzi [x 0.1 ohm] - např. 12255 = 1225.5Ω
Výstup :	**teplota_Kelvin** hodnota teploty v Kelvinech [x 0.1 K] - např. 2932 = 293.2K (+20.0°C)

Odpor je v 0.1 ohm, tedy hodnota 10000 = 1000.0 ohm. Výstupní hodnota je v 0.1K (desetiny Kelvina).

R2Ni1k6180

Funkce pro převod hodnoty odporu čidla Ni1000/6180ppm na teplotu.

Deklarace :	**function word R2Ni1k6180 (word odpor)**
Parametr 1 :	**odpor** hodnota odporu pro konverzi [x 0.1 ohm] - např. 12255 = 1225.5Ω
Výstup :	**teplota_Kelvin** hodnota teploty v Kelvinech [x 0.1 K] - např. 2932 = 293.2K (+20.0°C)

Odpor je v 0.1 ohm, tedy hodnota 10000 = 1000.0 ohm. Výstupní hodnota je v 0.1K (desetiny Kelvina).

Matematické funkce

Nad rámec běžné aritmetiky ve výrazech je možno použít i složitější matematické operace, které nabízí PLC řady 400 ve formě vestavěných funkcí. Pracují převážně s datovým typem float. Pro informaci, zda matematická operace dopadla bez chyby nebo s chybou je vyhrazen systémový bit

MATH_ERROR. Matematické funkce tento bit nastavují v případě chyb. V případě, že operace dopadne bez chyby, není hodnota bitu ovlivněna. Mazání chybového bitu je ponecháno na uživateli.

Deg2Rad

Funkce převádí hodnotu zadanou v desetinách úhlového stupně na hodnotu radiánu tj. hodnotu v rozsahu 0.0 – n*360.0° převede na hodnotu v rozsahu 0 – n*2π. Pokud zadáme zápornou hodnotu úhlových stupňů, chápe úhel převodní funkce ve směru běhu hodinových ručiček, kladné hodnoty chápe proti směru hodinových ručiček.

Deklarace :	**function float Deg2Rad (int deg)**	
Parametr 1 :	**deg**	hodnota úhlu v desetinách stupně [x 0.1°] - např. 905 = 90.5°
Výstup :	**rad**	hodnota úhlu v radiánech

MATH_ERROR: neovlivňuje

Rad2Deg

Funkce převádí hodnotu úhlu v rozsahu 0 – n*2π na hodnotu desetin stupně v rozsahu 0.0 – 360.0°. Funkce vrací vždy kladnou hodnotu tj. chápe úhel proti směru hodinových ručiček

Deklarace :	**function word Rad2Deg (float rad)**	
Parametr 1 :	**rad**	hodnota úhlu v radiánech
Výstup :	**deg**	hodnota úhlu v desetinách stupně [x 0.1°] - např. 905 = 90.5°

MATH_ERROR: neovlivňuje

Sqrt

Funkce vrací druhou odmocninu zadaného čísla.

Deklarace :	**function float Sqrt (float a)**	
Parametr 1 :	**a**	hodnota (v plovoucí čárce)
Výstup :	**sqrt_a**	druhá odmocnina z a

MATH_ERROR: nastavuje pokud je a < 0

Sin

Funkce vrací hodnotu sinus ze vstupní proměnné.

Deklarace :	**function float Sin (float a)**	
Parametr 1 :	**a**	hodnota (v plovoucí čárce)
Výstup :	**sin_a**	hodnota sinus z a

MATH_ERROR: neovlivňuje

Cos

Funkce vrací hodnotu cosinus ze vstupní proměnné.

Deklarace :	**function float Cos (float a)**	
Parametr 1 :	**a**	hodnota (v plovoucí čárce)
Výstup :	**cos_a**	hodnota cosinus z a

MATH_ERROR: neovlivňuje

Tan

Funkce vrací hodnotu tangens ze vstupní proměnné.

Deklarace :	**function float Tan (float a)**	
Parametr 1 :	**a**	hodnota (v plovoucí čárce)
Výstup :	**tan_a**	hodnota tangens z a

MATH_ERROR: nastavuje při přetečení

ArcSin

Vrací hodnotu úhlu funkce sinus v radiánech.

Deklarace :	**function float ArcSin (float a)**	
Parametr 1 :	**a**	hodnota (v plovoucí čárce)
Výstup :	**arcsin_a**	hodnota úhlu v radiánech, příslušná k sinus=a

MATH_ERROR: nastavuje při a > 1

ArcCos

Vrací hodnotu úhlu funkce cosinus v radiánech.

Deklarace :	**function float ArcCos (float a)**	
Parametr 1 :	**a**	hodnota (v plovoucí čárce)
Výstup :	**arccos_a**	hodnota úhlu v radiánech, příslušná ke cosinus=a

MATH_ERROR: nastavuje při a > 1

ArcTan

Vrací hodnotu úhlu funkce tangens v radiánech.

Deklarace :	**function float ArcTan (float a)**	
Parametr 1 :	**a**	hodnota (v plovoucí čárce)
Výstup :	**arctan_a**	hodnota úhlu v radiánech, příslušná k tangens=a

MATH_ERROR: neovlivňuje

Ln

Vrací hodnotu přirozeného logaritmu

Deklarace :	**function float Ln (float a)**	
Parametr 1 :	**a**	hodnota (v plovoucí čárce)
Výstup :	**ln_a**	hodnota přirozeného logaritmu argumentu a

MATH_ERROR: nastavuje při a <= 0

Pow

Vrací hodnotu mocniny x^y

Deklarace :	**function float Pow (float x, float y)**	
Parametr 1 :	**x**	základ mocniny (mocněnec)
2 :	**y**	exponent (mocnitel)
Výstup :	**x^y**	hodnota mocniny **x^y**

MATH_ERROR: nastavuje při přetečení a podtečení

Exp

Vrací hodnotu exponenciální funkce, tj. mocniny čísla e.

Deklarace :	**function float Exp (float x)**	
Parametr 1 :	**x**	exponent (mocnitel)
Výstup :	**e^x**	hodnota mocniny **e^x**

MATH_ERROR: nastavuje při přetečení a podtečení

Ceil

Vrací nejmenší celočíselnou hodnotu, která je větší nebo rovna a

Deklarace :	**function float Ceil (float a)**	
Parametr 1 :	**a**	argument funkce
Výstup :	**ceil_a**	celočíselný strop

MATH_ERROR: neovlivňuje

Příklad:

```
Ceil(2.6)   ->     3.0
Ceil(-2.6)  ->    -2.0
```

Floor

Vrací největší celočíselnou hodnotu, která je menší nebo rovna a

Deklarace :	**function float Floor (float a)**	
Parametr 1 :	**a**	argument funkce
Výstup :	**floor_a**	celočíselný základ

MATH_ERROR: neovlivňuje

Příklad:

```
Floor(2.6)   ->    2.0
Floor(-2.6)  ->   -3.0
```

FMod

Vrací zbytek typu float po dělení a/b

Deklarace :	**function float FMod (float a, float b)**	
Parametr 1 :	**a**	dělenec
2 :	**b**	dělitel
Výstup :	**mod_ab**	zbytek po celočíselném dělení

MATH_ERROR: nastavuje při nekonečné (přetečené) hodnotě a, či pro b = 0

Příklad:

```
FMod(10.0, 3.0)   ->    1.0
```

ModF

Vrací desetinnou část čísla c a proměnnou pint nastaví na celočíselnou část čísla c. Obě získané hodnoty mají totožná znaménka.

Deklarace :	**function float ModF (float c, var longint pint)**	
Parametr 1 :	**c**	vstupní hodnota
2 :	**pint**	odkaz na proměnnou, kam se uloží celočíselná část
Výstup :	**dec**	desetinná část (zbytek po odečtení celočíselné části)

MATH_ERROR: neovlivňuje

Příklad:

```
ModF(-10.7,pint)        ->    vrátí -0.7 a pint nastaví na  -10
```

Funkce pro indexový přístup k I/O

Pro běžné, standardní typy aplikací se nepoužívají.

Jsou využitelné zejména pro speciální aplikace s variabilním či parametrickým osazením počtů a typů I/O modulů v systému.

U řídicích systémů řady 400 byl od základu změněn systém přístupu na jednotlivé vstupy/výstupy zařízení. Ty jsou nyní rozčleněny na logické uzly, přičemž každý uzel reprezentuje jeden I/O modul na PLC MPC400 nebo na rozšiřujících jednotkách MEX400.

Každý typ I/O modulu má definován svůj datový typ ve formě datové struktury. Přístup na jednotlivé vstupy/výstupy se tak odlišuje právě podle typu použitého modulu.

Máme-li například na adrese 17 modul A a na adrese 18 modul D, pak výstupy Y0 na obou modulech zapneme pomocí zápisů:

```
ioa[17].y ? 0 = 1
iod[18].y ? 0 = 1
```

Zápisy jsou sice formálně stejné, ale přístup k výstupu Y0 na každém z modulů je vázán na specifickou datovou strukturu a není tudíž možné jednoduše místo jednoho typu modulu použít jiný typ (položka "y" může být u různých modulů různě dlouhá, nebo ve struktuře různě umístěná). Tedy pokud budeme např. k modulu **D** přistupovat zápisem "**ioa**[17].y?0=1", nemusí výstupy vůbec fungovat.

Pokud je však třeba postavit software pro řídicí systém např. ve formě variabilní stavebnice pro opakované nasazení, může být výhodné zadávat např. přiřazení vstupů/výstupů ve formě odkazů na příslušné adresy I/O modulů, typ I/O bodu (x, y, i, o) a pořadové číslo, resp. index I/O bodu. Typ I/O modulu ovšem nelze předat takto flexibilně a nutnost jeho zadání ve formě výběru příslušné struktury by pro kýžený univerzální zápis byla nepřekonatelná.

Tento problém řeší funkce pro indexový přístup na vstupy/výstupy automatů a periferií, které samy zjistí typ I/O modulu připojeného na dané adrese a nasměrují požadavek na správné místo do paměti. Pokud např. požadovaný I/O bod na aktuálně připojeném I/O modulu neexistuje, funkce pro zápis neprovádějí nic a funkce pro čtení vrací nulu. Tímto způsobem je možné např. nastavit výstup Y0 (nebo přečíst vstup X0) na libovolném připojeném zařízení EXbus (uzlu, I/O modulu), aniž bychom znali jeho typ a konfiguraci (pokud, samozřejmě, takový Y nebo X vůbec má).

Výsledek operace každé této funkce je po jejím skončení uložen do globální systémové proměnné **IORESULT**:

_iores_ok	= 0x00	→	zápis/čtení proběhl bez chyby
_iores_ioerr	= 0x01	→	index vstupu/výstupu je neplatný, I/O bod neexistuje
_iores_nderr	= 0x02	→	neplatná adresa uzlu (požadovaný uzel je nedostupný)
_iores_acerr	= 0x03	→	odepřena deklarace virtuálního uzlu, uzel již existuje

Parametry, používané u níže uvedených funkcí mají tento význam:

node	adresa uzlu, tedy příslušného I/O modulu, na sběrnici EXbus
io	index pořadí I/O bodu v daném uzlu (např.5 pro X5, nebo Y5, I5, O5..)
value	hodnota pro zápis I/O bodu, bit (digitální I/O) nebo word (analog I/O)

Indexovací funkce jsou deklarovány odděleně pro digitální vstupy, digitální výstupy, analogové vstupy a analogové výstupy.

IOReadX

Funkce vyčte z paměti stav digitálního vstupu na daném uzlu, vrací jej hodnotou typu bit.

Deklarace :	**function bit IOReadX (word node, byte io)**	
Parametr 1 :	**node**	adresa uzlu na sběrnici EXbus
2 :	**io**	index pořadí digitálního vstupu v daném uzlu
Výstup :	**x**	stav digitálního vstupu

Dle výsledku operace nastavuje informační proměnnou IORESULT.

IOWriteX

Zapíše hodnotu typu bit do obrazu příslušného digitálního vstupu v paměti.

Deklarace :	**subroutine IOWriteX (word node, byte io, bit value)**	
Parametr 1 :	**node**	adresa uzlu na sběrnici EXbus
2 :	**io**	index pořadí digitálního vstupu v daném uzlu
3 :	**value**	hodnota (bit) pro zápis do obrazu vstupu v paměti
Výstup :	- - -	

Má smysl jen, když je třeba simulovat hodnotu vstupu na simulátoru, nebo pozměnit/přepsat/vnutit hodnotu vstupu na reálném PLC (použít proceduru nejlépe na začátku hlavní smyčky programu).

Dle výsledku operace nastavuje informační proměnnou IORESULT.

IOReadY

Funkce vyčte z paměti hodnotu připravenou pro digitální výstup, vrací hodnotu typu bit.

Deklarace :	**function bit IOReadY (word node, byte io)**	
Parametr 1 :	**node**	adresa uzlu na sběrnici EXbus
2 :	**io**	index pořadí digitálního výstupu v daném uzlu
Výstup :	**y**	stav digitálního výstupu

Dle výsledku operace nastavuje informační proměnnou IORESULT.

IOWriteY

Zapíše hodnotu typu bit do paměti určené pro příslušný digitální výstup.

Deklarace :	**subroutine IOWriteY (word node, byte io, bit value)**	
Parametr 1 :	**node**	adresa uzlu na sběrnici EXbus
2 :	**io**	index pořadí digitálního výstupu v daném uzlu
3 :	**value**	hodnota (bit) pro zápis do výstupu v paměti
Výstup :	- - -	

Dle výsledku operace nastavuje informační proměnnou IORESULT.

Příklad:

Pomocí indexovaného přístupu bychom při realizaci výše zmíněného příkladu použili funkci pro indexovaný zápis digitálního výstupu, například takto:

```
IOWriteY(17,0,1)
IOWriteY(18,0,1)
```

Jak je vidět, lze tímto způsobem nastavit výstup Y0 na I/O modulu, aniž by byl dopředu dán jeho typ. Je tu ale samozřejmě riziko neúspěchu (např. při pokusu nastavit Y0=1 na modulu G, který žádné výstupy nemá) - pro tento případ je vhodné ještě otestovat proměnnou IORESULT, zda vše proběhlo bez chyby.

IOReadI

Funkce vyčte z paměti hodnotu analogového vstupu, vrací hodnotu typu word.

Deklarace :	**function word IOReadI (word node, byte io)**	
Parametr 1 :	**node**	adresa uzlu na sběrnici EXbus
2 :	**io**	index pořadí analogového vstupu v daném uzlu
Výstup :	**i**	stav analogového vstupu

Dle výsledku operace nastavuje informační proměnnou IORESULT.

IOWritel

Zapíše hodnotu typu word do obrazu příslušného analogového vstupu v paměti.

Deklarace :	**subroutine IOWritel (word node, byte io, word value)**	
Parametr 1 :	**node**	adresa uzlu na sběrnici EXbus
2 :	**io**	index pořadí analogového vstupu v daném uzlu
3 :	**value**	hodnota (word) pro zápis do registru analog.vstupu v paměti
Výstup :	- - -	

Má smysl pouze v případech, kdy je třeba simulovat hodnotu vstupu na simulátoru, nebo pozměnit/přepsat/vnutit hodnotu vstupu na reálném PLC (je vhodné použít proceduru hned na začátku hlavní smyčky programu).

IOReadO

Funkce vyčte z paměti hodnotu připravenou pro analogový výstup, vrací hodnotu typu word.

Deklarace :	**function word IOReadO (word node, byte io)**	
Parametr 1 :	**node**	adresa uzlu na sběrnici EXbus
2 :	**io**	index pořadí analogového výstupu v daném uzlu
Výstup :	**o**	stav analogového výstupu

IOWriteO

Zapíše hodnotu typu word do paměti určené pro příslušný analogový výstup.

Deklarace :	**subroutine IOWriteO (word node, byte io, word value)**	
Parametr 1 :	**node**	adresa uzlu na sběrnici EXbus
2 :	**io**	index pořadí analogového výstupu v daném uzlu
3 :	**value**	hodnota (word) pro zápis do registru analog.výstupu v paměti
Výstup :	- - -	

Doplňkové funkce I/O subsystému

Jedná se o funkce, které pro běžné využití a v běžných aplikacích nejsou třeba.

První z nich je ryze informační - dává přehled o zapojeném uzlu/zařízení/modulu na konkrétní adrese EXbus. Další dvě jsou již jen pro zcela speciální účely - umožňují definovat virtuální zařízení na sběrnici EXbus pro potřeby simulace nebo testování programu.

Všechny tyto funkce pracují se stejným datovým typem proměnné - datovou strukturou **_io_dsc** pro ukládání popisných informací k uzlu. Ta je již v základu předdefinována a má tento tvar:

```
type struct
    longword dt3116,    ; datový typ položky č. 16-31 pro vstup/výstup
    longword dt1500,    ; datový typ položky č. 0-15 pro vstup/výstup
    longword mskin,     ; bitová maska označující vstupy
    longword mskout,    ; bitová maska označující výstupy
    longword mskpin,    ; bitová maska označující fyz.I/O na modulu automatu
    byte    ndtype      ; typ I/O modulu (hodnoty: 0,1,2,3)
end _io_dsc
```

IOGetNodeDef

Funkce zapíše do předané datové struktury _io_dsc, popisná pole IO uzlu ze zadané adresy v parametru node. Detailní popis struktury viz výše.

Deklarace :	**subroutine IOGetNodeDef (word node, var _io_dsc info)**	
Parametr 1 :	**node**	adresa uzlu na sběrnici EXbus
2 :	**info**	odkaz na strukturu, kam se uloží veškeré informace o daném uzlu
Výstup :	- - -	

IODefVirtualNode

DEFINOVÁNÍ VIRTUÁLNÍCH UZLŮ K FUNKCÍM PRO INDEXOVÝ PŘÍSTUP

Funkce má dvě varianty - s umístěním popisné struktury v kódové (konstantní) paměti, nebo v datové paměti.

Deklarace :	**function bit IODefVirtualNode(word node, const _io_dsc info)** **function bit IODefVirtualNode (word node, var _io_dsc info)**	
Parametr 1 :	**node**	adresa uzlu na sběrnici EXbus
2 :	**info**	odkaz na strukturu, odkud funkce převezme kompletní popis uzlu
Výstup :	**ok**	0 = chyba, 1 = úspěch

Použití této doplňkové funkce přichází v úvahu tehdy, když se v softwaru využívají funkce pro indexový přístup ke vstupům/výstupům a potřebné I/O moduly nejsou k PLC fakticky připojeny. Pokud funkce indexového přístupu v softwaru využity nejsou, není třeba použít ani tuto funkci.

Aby mohly funkce pro indexovaný přístup ke vstupům/výstupům fungovat, je nezbytné, aby měly k dispozici informaci o typu a struktuře příslušného uzlu na dané adrese linky EXBUS. V případě reálné sítě tyto informace poskytuje "on-line" vždy příslušná periferie. Pokud však daná periferie resp. uzel neexistuje, musíme potřebné informace dodat. K tomu slouží systémová funkce IODefVirtualNode. Do funkce se předává proměnná typu **_io_dsc** (datová struktura, popis viz výše), která může být umístěna buď v datové nebo kódové paměti.

Za provozu platí absolutní přednost popisné datové struktury zjištěné z reálného zařízení před strukturou definovanou pomocí funkce IODefVirtualNode. Z toho plyne, že pokud nainstalujeme pro potřeby simulace např. virtuální I/O modul pomocí funkce IODefVirtualNode, není nutné programovou konstrukci pro použití v reálné síti vyřazovat. V reálné síti nalezené adresy aktivních uzlů automaticky přepíší odpovídající datový popis dle reálného stavu. A pokud funkce IODefVirtualNode zjistí, že na dané IO adrese již datový popis je, neprovede nic a nastaví na hodnotu **_iores_acerr** systémovou proměnnou **IORESULT** (viz výše u indexového přístupu).

Příklad definice virtuálního IO modulu A na adrese 16:

```
; definice datové struktury pro popis modulu A:
code _io_dsc modul_a = (
  0xFFFFFFFF,
  0xFFFFFFF0,
  0x00000001,
  0x00000002,
  0x00000003,
  0x01
)
; zápis v inicializační části hlavní smyčky programu PLC:
;---------------------------------------
if RESET then begin
  IODefVirtualNode(16,modul_a)          ; instalace virtuálního modulu A na adresu
16
end
```

Datové definice I/O modulů řady 400 pro potřeby funkce IODefVirtualNode:

```
;-------------------------- MODUL A
code _virtual_io modul_a = (
  0xFFFFFFFF,
  0xFFFFFFF0,
  0x00000001,
  0x00000002,
  0x00000003,
  0x01 )
;-------------------------- MODUL B nebo C
code _virtual_io modul_b = (
  0xFFFFFFFF,
  0xFFF05555,
  0x000000FF,
  0x00000300,
  0x000002FF,
  0x01 )
;-------------------------- MODUL D nebo E
code _virtual_io modul_d = (
  0xFFFFFFFF,
```

```
  0xFFFF0555,
  0x0000003F,
  0x000000C0,
  0x000000BF,
  0x01 )
;-------------------------- MODUL F
code _virtual_io modul_f = (
  0xFFFFFFFF,
  0xFFFFD554,
  0x00000001,
  0x0000007E,
  0x0000007F,
  0x01 )
;-------------------------- MODUL G
code _virtual_io modul_g = (
  0xFFFFFFFF,
  0xFFFFFFF0,
  0x00000003,
  0x00000000,
  0x00000003,
  0x01 )
;-------------------------- MODUL H nebo I
code _virtual_io modul_h = (
  0xFFFFFFFF,
  0xFFF05555,
  0x000002FF,
  0x00000100,
  0x000002FF,
  0x01 )
;-------------------------- MODUL K nebo J
code _virtual_io modul_k = (
  0xFFFFFFFF,
  0xFFFF0555,
  0x000000BF,
  0x00000040,
  0x000000BF,
  0x01 )
;-------------------------- MODUL L
code _virtual_io modul_l = (
  0xFFFFFFFF,
  0xF1555554,
  0x00002000,
  0x00001FFF,
  0x00000003,
  0x01 )
;-------------------------- MODUL M
code _virtual_io modul_l = (
  0xFFFFFFFF,
  0xF1555554,
  0x00000000,
  0x00003FFF,
  0x00000003,
  0x01 )
```

IOList, IOVirt

IOList a IOVirt nepředstavují funkce ani procedury. Jsou to pole příznaků, která označují přítomnost/nepřítomnost IO uzlu na lince EXBUS. Jedná se o pole bajtů předdefinované v systému takto:

var byte[128] iolist

var byte[128] iovirt

Každý z IO uzlů, které mohou být připojeny na lince EXBUS má vyhrazen jeden bit. Vztah mezi adresou a přiřazeným bitem je dán indexem do pole, který vypočítáme tak, že IO adresu dělíme osmi a číslem bitu, které odpovídá zbytku po dělení IO adresy osmi. Pokud daný bit nalezneme nastavený na 1, znamená to, že dotyčný IO uzel je na lince EXBUS připojen, že je aktivní a že popisná data ve struktuře _io_dsc jsou platná. Současně je možné používat popisované bity pro detekci výpadku zařízení popř. komunikace s ním na lince EXBUS. Pole iolist obsahuje všechny instalované (provozované) IO uzly. Pole iovirt informuje o tom, zda daný (provozovaný) IO uzel, je definován programově a je tudíž virtuální (IO zařízení není fyzicky připojeno).

Funkce vrstvy TEXT - textová obrazovka

Textová obrazovka je kompatibilní svým chováním i vestavěnými funkcemi se všemi dosavadními řadami PLC MICROPEL (K1, K10, MPC300 a částečně i MT201). Firmware řady MPC400 ale nabízí, krom těch stávajících standardních funkcí, několik novinek:

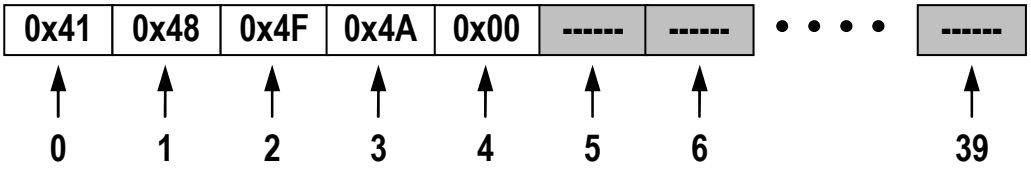
Nový datový typ dstring

Je zde nově definován datový typ dstring. Umožňuje práci s texty v datové paměti. Dosud bylo možné definovat text pouze typem "string" v kódové, tedy neměnné paměti. Protože nový typ je de-facto obyčejné pole bytů v datové paměti, umožňuje snadnou a variabilní přípravu textu v paměti po jednotlivých znacích a jeho následné vytištění standardní funkcí DISPLAY (do sady funkcí DISPLAY je nově přidaná funkce akceptující typ dstring).

Typ **dstring** je definován takto:

type byte[40] dstring

Ukončovací znak tisku představuje hodnota 0.



Příklad dle výše uvedeného obrázku:

Pokud chceme pomocí proměnné typu dstring vytisknout text „AHOJ“ musí v jednotlivých bajtech proměnné typu dstring být ASCII hodnoty znaků „AHOJ“ a bezprostředně za nimi ukončovač - bajt s

hodnotou 0. Na hodnotě bajtů následujících za ukončovačem již nezáleží a funkce tisku na displej je ignorují.

Virtuální displej/editor

Slouží zejména pro formátování a editaci hodnot v grafické vrstvě.

Aby se využily formátovací schopnosti funkcí DISPLAY např. při tisku různých datových typů s různým nastavením parametru FORMAT i v grafické vrstvě, je tu zavedena možnost výstupu funkcí DISPLAY nejen přímo na obrazovku displeje, ale i do paměti.

Pro tento případ je v datové paměti vyhrazen 2x speciální prostor o rozměru 1 řádku displeje (tedy 40B). Jeden je určen pro zobrazování, druhý pro editaci. Editační řádek se liší tím, že na aktuálním umístění kurzoru zobrazuje blikající čáru.

Výstup funkcí DISPLAY se směruje do paměti jednoduše nastavením POSITION = 10000 (zobrazovací řádek), nebo POSITION = 10040 (editační řádek). Poté je třeba zavolat funkci GdiDisplayText, která je speciálně určena k vytištění obsahu virtuálního řádku do grafické vrstvy. Blíže viz popis funkce GdiDisplayText.

Pozn.: Při používání editorů v grafické vrstvě je třeba počítat s pomalejší reakcí na stisk klávesy a to hlavně při větším množství grafických operací, kdy může být znatelná doba potřebná pro vykreslení celé obrazovky, zde se může při řešení odezvy dobře uplatnit bit GDI_KEEP.

Chod zobrazování textové obrazovky

Automaty řady 400 jsou vybaveny vylepšenou verzí virtuální textové obrazovky, která potlačuje jev problikávání při překreslování dat na obrazovce stylem „smažu/vytisknu“. U původního systému textové obrazovky (používán u řady MPC300 a K) bylo nutné pečlivě dbát na způsob překreslování jednotlivých údajů na obrazovce, aby se zabránilo problikávání překreslovaných textů. U nové verze textové obrazovky je její překreslování synchronizováno. Po každém doběhu uživatelské programové smyčky je aktuální obsah textové obrazovky zkopírován do vyrovnávací paměti. Tím je dosaženo toho, že v uživatelském programu je možné v průběhu programové smyčky libovolně mazat obrazovku a zase na ni tisknout až do požadovaného výsledku, který musí být hotov na konci běhu aktuální programové smyčky. Na konci smyčky se teprve virtuální textová obrazovka otiskne do vyrovnávací paměti a předá se k finálnímu tisku na displej. Celý displej se přetiskne 10x za sekundu. Teprve rychlejší střídání znaků než 5x za sekundu se může projevit jako nepravidelné blikání.

Pro tisk textů a hodnot proměnných je důležitá organizace textové vrstvy displeje a s ní spojené řídicí proměnné tisku POSITION a FORMAT. Z důvodu kompatibility je textová vrstva organizována do jednotlivých řádků s délkou 40 znaků. Pokud má daný automat displej s menším počtem znaků, jsou přebytečné znaky řádku odříznuty. V opačném případě tj. např. u MT470, nezabírají řádky textové vrstvy celou šířku displeje. Pozice pro tisk znaku je tedy dána 40x č.řádku + sloupec přičemž maximální hodnota pro sloupec je 39. Maximální číslo řádku omezuje svislý rozměr displeje automatu a hodnota POSITION = 10000, určená pro virtuální řádek a 10040 pro editor. Pro svislý rozměr textové vrstvy je tak k dispozici max.250 řádků.

Vestavěná funkce Display je klíčovou funkcí pro tisk do textové vrstvy. Funkce pracuje s přetíženým parametrem a na základě datového typu tohoto parametru a hodnoty proměnné FORMAT a POSITION provede tisk parametru do finálního tvaru na textový řádek.

Display

Tiskne obsah řetězce znaků datového typu dstring na obrazovku displeje. Počáteční pozici tisku určuje hodnota proměnné POSITION a tato pozice se posouvá o tolik znaků, kolik jich bylo vytisknuto. Po provedeném tisku na displej tedy ukazuje proměnná POSITION na další volný znak.

Deklarace :	**subroutine Display (var dstring str)** **subroutine Display (const string str)**
Parametr :	**str -** max. 40-znakový textový řetězec k tisku na displej, uložený v datové paměti dstring nebo kódové paměti const string
Výstup :	- - -

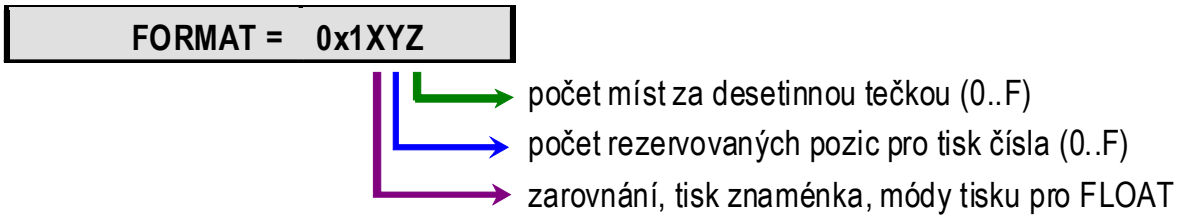
Display

Tiskne hodnotu proměnné. Formát tisku je řízen proměnnou FORMAT, pozici tiskového výstupu nastavuje proměnná POSITION.

Deklarace :	**subroutine Display (byte data)** **subroutine Display (word data)** **subroutine Display (int data)** **subroutine Display (longword data)** **subroutine Display (longint data)** **subroutine Display (float data)**
Parametr :	**data –** hodnota pro formátovaný tisk na řádek textové vrstvy
Výstup :	- - -

Na tomto místě je vhodné upozornit na záludnost použití univerzálního formátu 0 ze starší verze jazyka Simple 2. S tímto formátem tisknou zobrazovací funkce „Display“ hodnotu proměnné na právě potřebný počet znaků. To znamená, že pokud bez jakéhokoli opatření tiskneme proměnnou s hodnotou 100 a tato hodnota se v čase změní na 99, bude na displeji zobrazeno 990 a ne 99, jak by se na první pohled zdálo. Přebytečná 0 zůstala zobrazena z předešlého tisku hodnoty 100. Pro tisk proměnných tedy doporučujeme důsledně používat formátování tisku pomocí zarovnání a vyhrazení tiskových pozic nastavením vhodné hodnoty proměnné FORMAT. Tato proměnná je typu word a jejích 16 bitů můžeme rozdělit po čtveřicích bitů na čtyři pole. To je důležité proto, že pomocí hodnot těchto polí řídíme vybrané parametry tisku. Formátování podle Simple 4 shrnuje následující tabulka. Jednotlivá pole tak můžeme pomocí hexadecimálního zápisu zapisovat s použitím hodnot 0-9 a A-F.

Ze starší verze jazyka má smysl používat pouze formát 120 (0x0078) - tisk celočíselné hodnoty jako ASCII kódu, formát pro tisk jednotlivých písmen řetězce.



Pole X	Zarovnání	Znaménko "+"	Tisk proměnných FLOAT	Tisk celočíselných proměnných
0	vlevo	ne	AUTO - tisk buď v klasickém tvaru (±XXX.XXXX), nebo v exponenciálním (±X.XXXE±XX). Pole Z udává počet platných míst	DECI standardní dekadický tvar
1	vpravo	ne		
2	vlevo	ano		
3	vpravo	ano		
4	vlevo	ne	FIX - tisk s pevným počtem desetin. Tisk čísla je ve tvaru ±XXX.XXXX. Pole Z udává počet desetinných míst	HEX tisk čísla v hexadecimálním tvaru (desetiny se ignorují)
5	vpravo	ne		
6	vlevo	ano		
7	vpravo	ano		
8	vlevo	ne	EXP - exponenciální vyjádření. Tisk čísla ve tvaru ±X.XXXE±XX, pole Z = počet desetinných míst, exponent je dvoumístný	DECI
9	vpravo	ne		0-DECI, **předsaz. nuly**
A	vlevo	ano		DECI
B	vpravo	ano		DECI
C	vlevo	ne		HEX
D	vpravo	ne		0-HEX, **předsaz. nuly**
E	vlevo	ano		HEX
F	vpravo	ano		HEX

DisplayClear

Maže textovou obrazovku displeje tak, že do všech pozic textové vrstvy zapíše průhledný znak " " (mezera).

Deklarace :	**subroutine DisplayClear ( )**
Parametr :	- - -
Výstup :	- - -

U řady 400 je díky synchronizaci překreslování textové obrazovky na konec hlavní smyčky nyní možné i často mazat celou obrazovku a hned následně na ni tisknout. Pokud se vše stihne během jednoho běhu smyčky, nedochází k žádným rušivým jevům.

DisplaySize

Vrací rozměr textové obrazovky pro daný automat.

Deklarace :	**subroutine DisplaySize (var pxy size)**
Parametr :	**size** odkaz na proměnnou pxy, kam uloží rozměr textové obrazovky
Výstup :	- - -

Do předané proměnné size vloží rozměr textové obrazovky, který je na daném PLC k dispozici:

size.x = horizontální rozměr textové obrazovky (počet znaků)

size.y = vertikální rozměr textové obrazovky (počet řádků)

Protože na různých PLC mohou být displeje různé velikosti, různého rozlišení i s různým počtem znaků textové obrazovky, přináší tato funkce možnost automatického přizpůsobení softwaru různým modelům a typům PLC. Využívá se hlavně v podpůrných knihovnách.

Pozn.: struktura pxy je obsažená již v základních definicích překladače pro MPC400 a má význam obecně použitelné pravoúhlé souřadnice x-y, viz kap. 5.

DisplayFColor

Nastaví barvu písma, která bude platit pro všechny výstupy na textovou obrazovku funkcemi DISPLAY (až do další změny).

Deklarace :	**subroutine DisplayFColor (byte fcolor)**
Parametr :	**fcolor** *(foreground color)* 8-bitová barva popředí (písma)
Výstup :	- - -

Barva je dána 8-bitovým kódem, blíže viz stať BARVY. Důležité barvy jsou označeny předdefinovanými konstantami. Průhledná (transparentní) barva má kód 255.

DisplayBColor

Nastaví barvu pozadí písma, která bude platit pro všechny výstupy na textovou obrazovku funkcemi DISPLAY (až do další změny).

Deklarace :	**subroutine DisplayBColor (byte bcolor)**
Parametr :	**bcolor** *(background color)* 8-bitová barva pozadí
Výstup :	- - -

Barva je dána 8-bitovým kódem, blíže viz stať BARVY. Důležité barvy jsou označeny předdefinovanými konstantami. Průhledná (transparentní) barva má kód 255.

DisplayGetFColor

Vrátí 8-bitový kód barvy písma (popředí), která je právě aktuálně používaná pro tisk na textovou obrazovku funkcemi DISPLAY. Blíže k barvám a jejich kódům viz stať BARVY.

Deklarace :	**function byte DisplayGetFColor ( )**
Parametr :	- - -
Výstup :	**fcolor** aktuálně nastavená 8-bitová barva popředí (písma)

DisplayGetBColor

Vrátí 8-bitový kód barvy pozadí, která je právě aktuálně používaná pro tisk na textovou obrazovku funkcemi DISPLAY. Blíže k barvám a jejich kódům viz stať BARVY.

Deklarace :	**function byte DisplayGetBColor ( )**
Parametr :	- - -
Výstup :	**bcolor** aktuálně nastavená 8-bitová barva pozadí

DisplaySetFont

Nastaví uživatelský font pro textovou vrstvu displeje a je náhradou za možnost definovat uživatelské znaky u starších řad automatů. POZOR ! Datová struktura musí obsahovat definici fontu ve velikosti, která odpovídá výchozímu systémovému fontu (pro MPC400 má systémový font rozměr 8x15, MT470 pak 15x30). Varianta funkce bez parametru, nastavuje výchozí systémový font, který má kódování CE.

Deklarace :	**subroutine DisplaySetFont (const font p)** **subroutine DisplaySetFont ()**
Parametr :	**p** datová struktura s definici uživatelského fontu
Výstup :	- - -

DisplaySetSize

Nastaví velikost textové vrstvy obrazovky. Velikost se udává v počtu znaků, kdy položka x struktury pxy odpovídá vodorovnému počtu znaků a položka y pak počtu řádek. Funkce najde uplatnění především u typů s větším displejem např. MT470. Varianta bez parametru, vrací nastavení textové vrstvy displeje na výchozí rozměr automatu.

Deklarace :	**subroutine DisplaySetSize (var pxy sz)** **subroutine DisplaySetSize (const pxy sz)** **subroutine DisplaySetSize ()**
Parametr :	**sz** velikost textové obrazovky zadaná ve znacích systémového fontu
Výstup :	- - -

DisplaySetOrg

Nastaví počáteční souřadnice levého horního rohu textové obrazovky. Souřadnice se udává v počtu znaků, kdy položka x struktury pxy odpovídá vodorovnému počtu znaků fyzické obrazovky automatu a položka y pak počtu řádek. Funkce najde uplatnění především u typů s větším displejem např. MT470.

Deklarace :	**subroutine DisplaySetOrg (var pxy org)** **subroutine DisplaySetOrg (const pxy org)**
Parametr :	**org** souřadnice počátečního znaku textové obrazovky
Výstup :	- - -

```
; nastavení textové obrazovky do pravého dolního rohu
; obrazovka bude zabírat ¼ celkové plochy fyzické obrazovky MPC400
var pxy org
var pxy sz
if reset then begin
org.x = 13
org.y =  3
sz.x = 12
sz.y = 3
DisplaySetSize(sz)
DisplaySetOrg(org)
end
```

DisplayCursorCode

Umožňuje volbu znaku, který budou systémové editační funkce používat v textové obrazovce. Tato funkce je užitečná v případě, že standardně použitý znak je v nastaveném kódování (systémovém fontu) použit k jiným účelům či jako specifický znak jazyka (viz. případ kódu 255 v azbuce).

Deklarace :	**subroutine DisplayCursorCode (byte code)**
Parametr :	**code** kód znaku použitý pro vestavěné systémové editační funkce
Výstup :	- - -

DisplayEnable

Vypíná formátování a zobrazení textové vrstvy do finálního zobrazení na fyzickém displeji automatu. Pokud textovou vrstu v aplikačním programu nepoužíváme, je vhodné její zobrazení vypnout, protože se tím uspoří výpočetní výkon použitý pro systémové služby a tento výkon je pak k dispozici uživatelskému programu.

Deklarace :	**subroutine DisplayEnable (bit enable)**	
Parametr :	**enable**	0 – textová vrstva se nepoužívá, 1 – textová vrstva se používá
Výstup :	- - -	

IsDisplayEnable

Zjišťuje aktuální stav zobrazování textové vrstvy displeje.

Deklarace :	**function bit IsDisplayEnable ()**	
Parametr :	- - -	
Výstup :	**ok**	1 = textová vrstva se zobrazuje, 0 = textová vrstva je vypnuta

DisplayGetLine

Umožňuje vyčíst zvolený řádek textové vrstvy displeje a to včetně virtuálních řádků 10000 10040.

Deklarace :	**subroutine DisplayGetLine (var dstring text, word position)**	
Parametr 1:	**text**	textová proměnná na uložení řádku displeje
2:	**position**	pozice od níž se text vyčte
Výstup :	- - -	

Počátek vyčtení je dán hodnotou **position** pro kterou platí:

position = 40 * ln + col,

kde **ln** značí číslo řádku a **col** číslo sloupce v daném řádku

Vrstva GRAPHIC

Soubor mnoha funkcí pro práci s grafickou vrstvou - kreslení základních objektů, zobrazování bitmap, nastavení barev, definice fontů, tisk textů a různé podpůrné a doplňkové funkce. Pro přehlednost jsou názvy všech funkcí týkajících se této vrstvy uvozeny písmeny "**Gdi**". Celá vrstva GRAPHIC pracuje s 8-bitovými barvami.

Funkce související s obsluhou vrstvy GRAPHIC

GdiFunctionTime

Funkce vrátí počet mikrosekund, po který trvala poslední grafická operace (volání grafické funkce typu **Gdi**)

Deklarace :	**function word GdiFunctionTime ( )**
Parametr :	- - -
Výstup :	**time** délka trvání poslední volané grafické funkce v mikrosekundách [µs]

GdiThreadTime

Funkce vrátí počet mikrosekund, který souhrnně zabraly všechny grafické operace (volání grafických funkcí typu **Gdi**) v aktuálním průchodu, tedy od začátku hlavní programové smyčky do místa volání této funkce.

Deklarace :	**function longword GdiThreadTime ( )**
Parametr :	- - -
Výstup :	**time** délka trvání všech grafických funkcí ve smyčce v mikrosekundách [µs]

GdiGraphSize

Do předané proměnné size (struktura typu pxy) vloží x-y rozměr grafické obrazovky (v pixelech), který je na daném PLC k dispozici:

size.x = horizontální rozměr (počet pixelů)

size.y = vertikální rozměr (počet pixelů)

Deklarace :	**subroutine GdiGraphSize (var pxy size)**
Parametr :	**size** odkaz na proměnnou pxy, kam uloží rozměr grafické obrazovky (v pixelech)
Výstup :	- - -

Protože se v portfoliu stávajících i budoucích PLC MICROPEL mohou vyskytovat typy s displeji různé velikosti a různého rozlišení, přináší tato funkce možnost automatického přizpůsobení softwaru různým modelům a typům PLC.

Pozn.: blíže ke struktuře pxy viz kap. 5.

GdiWindow

Nastavuje na displeji okno, kam budou kreslit grafické funkce Gdi. Mimo tento zadaný výřez se žádný výstup do grafické vrstvy neprovede, veškerý grafický výstup funkcí Gdi.. bude oříznut tímto obdélníkem. Existuje ve více variantách - dle typu a umístění parametrů.

Deklarace :	**subroutine GdiWindow (var pxy tl, var pxy br)** **subroutine GdiWindow (const pxy tl, var pxy br)** **subroutine GdiWindow (var pxy tl, const pxy br)** **subroutine GdiWindow (const pxy tl, const pxy br)**
Parametr 1 :	**tl** *(top left corner)* souřadnice levého horního rohu okna (v pixelech)
2 :	**br** *(bottom right corner)* souřadnice pravého dolního rohu (v pixelech)
Výstup :	- - -

Varianta bez parametrů

VRACÍ OŘEZOVÉ OKNO DO VÝCHOZÍHO STAVU

tj. na plný rozměr grafické obrazovky a ořezové okno se tím de-facto ruší.

Deklarace :	**subroutine GdiWindow ( )**
Parametr :	- - -
Výstup :	- - -

GRAPHIC - práce s barvami

GdiSetUserColor

Umožňuje doplnit do palety 256 barev uživatelské barvy v té části palety, která je pro tento účel vyhrazena (dovolené indexy barev jsou 237....252, celkem 16 barev). Zadání indexu mimo tento rozsah funkce ignoruje a neprovádí nic. Blíže k barvám viz stať BARVY.

Deklarace :	**subroutine GdiSetUserColor (byte index, word color)**
Parametr 1 :	**index** 8-bitová barva (index do palety barev)
2 :	**color** 16-bitová barva (v kódování RGB 5:6:5)
Výstup :	- - -

Pozn.: 16-bitová barva s kódem 0x07E0 v systému RGB5:6:5 je chápána jako **průhledná**!

GdiFColor

Nastaví 8-bitovou barvu popředí - kreslicího nástroje (pera) nebo písma, která bude platit pro všechny výstupy na grafickou obrazovku funkcemi Gdi, až do další změny.

Deklarace :	**subroutine GdiFColor (byte fcolor)**	
Parametr :	**fcolor**	*(foreground color)* 8-bitová barva popředí (písma, kreslicího pera)
Výstup :	- - -	

GdiBColor

Nastaví 8-bitovou barvu pozadí písma, která bude platit pro všechny výstupy na grafickou obrazovku funkcemi Gdi, až do další změny.

Deklarace :	**subroutine GdiBColor (byte bcolor)**	
Parametr :	**bcolor**	*(background color)* 8-bitová barva pozadí
Výstup :	- - -	

GdiGetFColor

Vrátí 8-bitový kód barvy popředí (písma, kreslicího pera), která je právě aktuálně používaná pro tisk na grafickou obrazovku funkcemi Gdi. Blíže k barvám a jejich kódům viz stať BARVY.

Deklarace :	**function byte GdiGetFColor ( )**	
Parametr :	- - -	
Výstup :	**fcolor**	aktuálně nastavená 8-bitová barva popředí (písma, kreslicího pera)

GdiGetBColor

Vrátí 8-bitový kód barvy pozadí, která je právě aktuálně používaná pro tisk na grafickou obrazovku funkcemi Gdi. Blíže k barvám a jejich kódům viz stať BARVY.

Deklarace :	**function byte GdiGetBColor**	
Parametr :	- - -	
Výstup :	**bcolor**	aktuálně nastavená 8-bitová barva pozadí

GdiFillScr

Podprogram vyplní celou plochu grafické vrstvy GRAPHIC zadanou 8-bitovou barvou.

Deklarace :	**subroutine GdiFillScr (byte color)**
Parametr :	**color** 8-bitová barva pro vyplnění obrazovky
Výstup :	- - -

Jedná se de-facto o smazání grafické obrazovky (vrstvy GRAPHIC) zvolenou barvou. Je vhodné touto funkcí vždy začít nový cyklus vykreslení grafické obrazovky, pokud potřebujeme vyplnit obrazovku jinou, než průhlednou barvou (vyplnění průhlednou barvou zajišťuje automaticky firmware automatu).

Pozn.: Při vyplnění průhlednou barvou se zviditelní vrstva BACKGROUND, která je umístěna "nejníže" v hierarchii vrstev MGD.

GRAPHIC - práce s textem

GdiSetFont

Podprogram nastavuje aktuální font (písmo, soubor znaků) pro tisk textu v grafické vrstvě. Nastavení je platné do další změny. Parametrem podprogramu je struktura typu **font**, uložená výhradně v programové (konstantní) paměti. Tuto konstantní strukturu automaticky generuje nástroj pro začlenění fontů do programu. Její název pak reprezentuje daný font v uživatelském programu (zadává se jako parametr funkce GdiSetFont). Popis struktury **font** viz kapitola 5.

Deklarace :	**subroutine GdiSetFont (const font fnt)**
Parametr :	**fnt** odkaz na definiční strukturu fontu
Výstup :	- - -

Pozn.: Ve výchozím stavu po zapnutí PLC je pro funkce Gdi nastaven vestavěný systémový font, stejný jaký se používá v textové vrstvě: neproporcionální o rozměru 8x15 pixelů.

Varianta bez parametrů

VRACÍ NASTAVENÍ FONTU DO VÝCHOZÍHO STAVU

tj. na systémový font 8x15 pixelů, který je nastaven po zapnutí nebo resetování PLC.

Deklarace :	**subroutine GdiSetFont ( )**
Výstup :	- - -

Poznámka ke konstrukci fontu:

Má smysl jen pro ty, kteří budou font tvořit ručně ve zdrojovém textu (ano, i to je možné), nebo jiným způsobem, svým vlastním softwarem apod. Kdo použije pro vytváření a generování fontů podpůrný software MICROPEL, nemusí se následujícími detaily zabývat.

Položka **ptr** ve struktuře **font** odkazuje na datový typ string (řetězec) v konstantní paměti, kde jsou v jedné řadě za sebou uložena data fontu.

Musí zde být vždy úplná sada 256 znaků. Znaky jsou ukládány po bajtech tak, že pro každý řádek znaku je použit celistvý počet bajtů. Pro font s maximální šířkou znaku 1-8 bodů je řádek znaku uložen v 1B. Písma s maximální šířkou znaku od 9 do 16 bodů mají uložen jeden řádek znaku ve 2B. Maximální šířku udává položka **width** a maximální výšku položka **height** ve struktuře font. Tím je zároveň dáno, kolik B paměti zabere každý z 256 znaků fontu (ještě je třeba přidat 1B na aktuální šířku znaku **W** - viz dále).

Jednotlivé pixely znaku jsou ukládány zprava doleva od bitu b0 po bit bN, kde N je max. šířka písma daná položkou **width**. Zde je znak ukládán bajt po bajtu ve formátu: **W** + data, kde **W** udává aktuální šířku znaku, následovaný zakódovanými řádky znaku dle výše popsaného způsobu.

Jak vidno, font v grafické vrstvě může být i tzv. proporcionální, kdy každý znak může mít různě nastavenou šířku podle toho jaký má tvar a písmena se pak tisknou těsně k sobě bez rušivých mezer okolo úzkých znaků. U textu vytištěného takovým fontem je však obtížné snadno určit jeho délku v pixelech, proto jsou definovány i funkce pro zjištění rozměru textu.

Naproti tomu font v textové vrstvě je pouze neproporcionální, protože každý znak je umísťován na přesnou pozici v pevném rastru.

Příklad zápisu fontu o velikosti 5x7 znaků:

Nejprve je definován řetězec představující data jednotlivých znaků. Je zde užito zápisu pro slučování textových řetězců pomocí znaku + (z důvodu omezené šířky řádku zdrojového textu i z důvodu lepší přehlednosti). Každý řádek zápisu představuje jeden znak fontu. Je vždy uvozen hodnotou 5, která specifikuje šířku znaku. Protože jsou všechny znaky stejně široké, jedná se o font neproporcionální.

```
;------------------------------------ DATA FONTU
code string font_5x7_data = (
"\005\000\000\000\000\000\000\000" +     ; znak 0
"\005\000\000\000\000\000\000\000" +     ; znak 1
                                         ; .........
"\005\000\000\000\000\000\000\000"       ; znak 255
)
;------------------------------------ definice fontu font_5x7 strukturou font
code font font_5x7 = (                   ;
  5,                                     ; width - max.šířka = 5pix
  7,                                     ; height - výška = 7pix
  font_5x7_data                          ; odkaz na data fontu ptr
)
;------------------------------------ použití fontu font_5x7 v programu
GdiSetFont(font_5x7)
```

GdiTextRect

Podprogram nastavuje ořezový obdélník pro tisk textu do grafické vrstvy. Ořezový obdélník se zadává parametrem **size** typu **pxy** (blíže ke struktuře **pxy** viz DATOVÉ STRUKTURY).

Zadaný ořezový obdélník textu představuje aktuální výsek pro tisk textu v grafické vrstvě a tvoří tak tzv. textový objekt. Tisk textu je vymezen tímto obdélníkem a provádí se pouze do něj a vůči tomuto obdélníku se provádí i zarovnávání textu (nastavení zarovnávání viz GdiTextMode).

Deklarace :	**subroutine GdiTextRect (var pxy size)** **subroutine GdiTextRect (const pxy sz)**
Parametr :	**size** odkaz na strukturu x-y s rozměrem ořezového obdélníku (v pixelech)
Výstup :	- - -

Pozn.: Nezávisle na finální velikosti textu je celý ořezový obdélník při tisku textu vyplněn barvou pozadí platnou pro Gdi funkce (nastavuje se funkcí GdiBColor).

TIP:

Aby byl obdélník správně nastaven vzhledem k zamýšlenému textu, je vhodné nejprve zjistit u připraveného textového řetězce jeho faktické rozměry při tisku konkrétním fontem (funkcemi GdiGetTextLen a GdiGetTextHeight).

GdiTextMode

Podprogram nastavuje mód tisku textu. Tento mód je při tisku používán až do další změny.

Deklarace :	**subroutine GdiTextMode (longword mode)**
Parametr :	**mode** parametry pro tisk textu
Výstup :	- - -

Parametr **mode** má následující význam (v bitovém vyjádření):

bit:	31 .. 9	8	7	6	5	4	3	2	1	0
název:	- - -	V1	V0	H1	H0	U	R	- - -	Z	- - -
význam:		vert.align		horiz.align		under.	rotate		zoom	

Natočení textu (rotate):

R=0 výchozí stav, bez otočení

R=1 otočení 90° doleva (proti směru hodinových ručiček)

Podtržení textu (underline):

U=0 výchozí stav, bez podtržení

U=1 znaky textu doplněny podtržením v dolní části

Vodorovné zarovnání (horizontal align), vzhledem k ořezovému obdélníku:

H1=0 H0=0 výchozí stav, zarovnání vlevo

H1=0 H0=1 zarovnání doprostřed

H1=1 H0=0 zarovnání vpravo

Svislé zarovnání (vertical align), vzhledem k ořezovému obdélníku:

V1=0 V0=0 výchozí stav, zarovnání dolů

V1=0 V0=1 zarovnání doprostřed

V1=1 V0=0 zarovnání nahoru

Zvětšení:

Z =0 výchozí stav, písmo ve standardní velikosti

Z=1 písmo zvětšené 2x

Pro zadání parametru mode lze použít i předdefinované konstanty:

Natočení o 90° do svislého směru:	**_vertical**	= 0x008
Podtržení:	**_underline**	= 0x010
Vodorovně vlevo:	**_left**	= 0x000
Vodorovně doprostřed:	**_centr**	= 0x020
Vodorovně vpravo:	**_right**	= 0x040
Svisle dolů:	**_bottom**	= 0x000
Svisle doprostřed:	**_vcentr**	= 0x080
Svisle nahoru:	**_top**	= 0x100
Zvětšení	**_zoom2**	= 0x002

Příklad nastavení pro podtržené písmo, zarovnané doprostřed ořezového boxu:

```
GdiTextMode(_underline + _centr + _vcetnr)
```

Natočení textu

Je podporován jen 1 stupeň otočení - o 90° vlevo. Při natočení textu je zároveň takto pootočeně chápán i ořezový obdélník a pootočí se i pojmy vlevo → dolů → vpravo → nahoru, takže při pohledu zepředu na otočený text např. zarovnání nahoru pošle text v rámci ořezového obdélníku k jeho levému kraji a zarovnání dolů jej pošle k pravému kraji.

Jinak řečeno, zarovnání je vždy chápáno vzhledem k logice tisknutého textu. Zvětšení 2x je chápáno jako zvětšení znaků fontu a nijak neovlivňuje nastavený ořezový obdélník.

GdiText

Zajišťuje tisk textu do grafické vrstvy. Existuje ve více variantách - dle typu a umístění parametrů.

Deklarace :	**subroutine GdiText (const string text, var pxy origin)**	
	subroutine GdiText (const string text, const pxy origin)	
	subroutine GdiText (var dstring text, var pxy origin)	
	subroutine GdiText (var dstring text, const pxy origin)	
	subroutine GdiText (const dstring text, var pxy origin)	
	subroutine GdiText (const dstring text, const pxy origin)	
Parametr 1 :	**text**	odkaz na textový řetězec typu string nebo dstring
2 :	**origin**	odkaz na souřadnice umístění textového objektu (v pixelech)
Výstup :	- - -	

Vytiskne tzv. textový objekt, což je obdélník s rozměry definovanými funkcí GdiTextRect. Obdélník je svým horním levým rohem umístěn na souřadnici danou parametrem **origin** a je celý vyplněn barvou pozadí (nastavenou funkcí GdiBColor). V tomto obdélníku je umístěn a zarovnán text podle nastavení daného funkcí GdiTextMode. Barva textu je dána barvou popředí (nastavenou funkcí GdiFColor). Parametr **origin** může být předán buď jako předpřipravená konstanta typu pxy, nebo jako proměnná typu pxy.

Parametr **text** může být předán jako konstanta rovnou ve volání funkce, nebo může být jako konstanta definován v kódové paměti, nebo může být předán ve formě pole znaků **dstring** (buď jako konstanta anebo jako proměnná v datové paměti).

Blokový text

Text může obsahovat i znaky pro nový řádek. Funkce tedy tiskne nejen řádek ale i blok textu. Cílový prostor tisku je dán nastaveným ořezovým obdélníkem a formátování pak nastaveným módem tisku. Víceřádkový řetězec pro tisk bloku textu zapíšeme např. takto:

```
code string ahojtext = "AHOJ\0x0D\0x0AJSEM TADY"
```

Hexadecimálně zapsané kódy znaků \0x0D\0x0A představují v ASCII kódování kombinaci CR/LF pro přechod na nový řádek.

GdiDisplayText

Podprogram tiskne virtuální řádky vyprodukované funkcemi Display (pro textovou vrstvu) do vrstvy grafiky. K dispozici jsou dva virtuální řádky o délce 40 znaků umístěné od pozice 10000 a 10040. Řádek od pozice 10000 je určen pro formátování tisku, řádek od pozice 10040 je určen pro editaci. Zobrazení obou řádků se tedy liší tím, že při tisku řádku 10040 se vytiskne krom textu ještě kurzor v podobě svislé čáry. To je proto, že v grafické vrstvě může být použit proporcionální font a při tisku

plného znaku by se musela šířka blikajícího kurzoru měnit podle aktuální šířky editované číslice či znaku.

Deklarace :	**subroutine GdiDisplayText (var pxy origin)** **subroutine GdiDisplayText (const pxy origin)**	
Parametr :	**origin**	odkaz na souřadnice umístění textového objektu (v pixelech)
Výstup :	- - -	

Řádek, který se vytiskne určuje aktuální hodnota proměnné POSITION. Pokud bude v rozsahu 10000-10039 tiskne řádek od pozice 10000 do aktuální pozice dané hodnotou proměnné POSITION. Obdobně se tiskne řádek editační pokud je hodnota POSITION od 10040.

GdiGetTextLen

Funkce vrací délku (v pixelech), která je potřebná pro tisk řetězce aktuálně zvoleným fontem.

Deklarace :	**function int GdiGetTextLen (const string text)** **function int GdiGetTextLen (const dstring text)** **function int GdiGetTextLen (var dstring text)**	
Parametr 1 :	**text**	odkaz na textový řetězec typu string nebo dstring
Výstup :	**length**	délka tisku textu (v pixelech)

Pokud je zadán text obsahující více řádků, vrací funkce maximální rozměr tj. hodnotu platnou pro nejdelší řádek. Hodnotu, získanou z této funkce, lze pak např. použít pro správné zadání rozměru do funkce GdiTextRect.

POZOR! U proporcionálního písma neplatí, že řádek s největším počtem znaků je zároveň nejdelší.

GdiGetTextHeight

Funkce vrací výšku (v pixelech), která je potřebná pro tisk řetězce aktuálně zvoleným fontem. Pokud je zadán text obsahující více řádků, bude funkce vracet odpovídající násobek výšky fontu. Hodnotu z této funkce lze pak např. použít pro správné zadání rozměru do funkce GdiTextRect.

Deklarace :	**function int GdiGetTextLen (const string text)** **function int GdiGetTextLen (const dstring text)** **function int GdiGetTextLen (var dstring text)**	
Parametr 1 :	**text**	odkaz na textový řetězec typu string nebo dstring
Výstup :	**height**	výška tisku textu (v pixelech)

GRAPHIC - práce s vektorovou grafikou

Zde jsou soustředěny funkce pro kreslení různých geometrických objektů a grafických prvků. Všechny zde uvedené funkce pracují pouze s barvou popředí (viz funkce GdiFColor).

Barva pozadí není využita, protože se kreslí vždy jen objekt bez pozadí. Pokud je třeba nakreslit např. vyplněný objekt s obrysovou čárou, kreslí se stejně nadvakrát - zvlášť čára a zvlášť výplň.

Soustava souřadnic

Měrnou jednotkou pro všechny rozměry a souřadnice je **pixel** (základní nejmenší dále nedělitelný obrazový bod). Protože se zobrazování děje na ploše displeje v 2-rozměrném prostoru, většina všech parametrů má význam 2D veličiny (rozměr x-y, souřadnice x-y...) v pravoúhlé souřadnicové soustavě. Pro zadávání pravoúhlých souřadnic a rozměrů se používá strukturovaný datový typ **pxy** (viz stať DATOVÉ STRUKTURY), obsahující položky x, y typu int. Díky tomu umožňuje též zadávání relativních (kladných i záporných) souřadnic vztažených k bodu 0:0.

Souřadnicová soustava displeje má střed (počátek) v levém horním rohu. Souřadnice y vzrůstá směrem dolů a souřadnice x vzrůstá směrem doprava.

Umísťování a tvorba objektů

Do všech kreslicích funkcí se vždy jako jeden ze základních parametrů předává **origin** (typ pxy) a znamená souřadnici umístění grafického objektu. Vlastní tvar nebo orientace objektu se vždy určuje ostatními parametry a je relativní vzhledem k souřadnici 0:0 (nevadí tedy, když některé souřadnice definující objekt budou i záporné). Tvar objektu potom stačí zadat jen 1x, nebo ho mít jako konstantu v kódové paměti a jen pouhou změnou umístění (parametr **origin**) lze buď vytvářet více identických objektů na obrazovce, nebo třeba plynule s objektem pohybovat.

GdiPenType

Nastavení parametrů kreslicího pera pro všechny níže uvedené vektorové kreslicí funkce.

Deklarace :	**subroutine GdiPenType (byte width, byte ptyp)**	
Parametr 1 :	**width**	šířka stopy pera (v pixelech)
2 :	**ptyp**	zatím nevyužito, zadávat hodnotu 0
Výstup :	- - -	

Podprogram nastavuje šířku stopy pera. Střed pera je ve středu uvnitř stopy, přičemž stopa má v mezích možností kruhový tvar. Šířku stopy je možné volit od 1 do 7. Při volbě 0 je použita stopa o rozměru 1 pixel. Parametr ptyp je v současné verzi grafického rozhraní nevyužit a zadává se do něj hodnota 0.

Pero se používá pro kreslení u všech kreslicích funkcí, kromě funkcí vyplňovacích.

GdiPoint

Nakreslí kruhový bod na zadaném umístění. Šířka bodu je dána parametry pera (viz GdiPenType).

Deklarace :	**subroutine GdiPoint (var pxy origin)** **subroutine GdiPoint (const pxy origin)**
Parametr :	**origin** odkaz na souřadnici umístění bodu (v pixelech)
Výstup :	- - -

GdiLine

Existuje ve více variantách dle typu a umístění parametrů. Vykreslí úsečku z počátečního bodu daného parametrem **origin**. Délka a orientace úsečky je dána parametrem **vector** (zde je možno využít toho že složky x a y datového typu pxy jsou typu int, tedy znaménkové a výsledný vektor tak může mířit na libovolnou stranu). Funkce tiskne tak, že k souřadnicím origin přičte souřadnice vector a tím získá druhý bod úsečky. Výhodou tohoto mechanismu je to, že s jednou nastaveným vektorem lze vytisknout vedle sebe více úseček nebo úsečkou po displeji pohybovat a to pouze změnou parametru origin. Kreslicí stopa je dána parametry pera (viz GdiPenType).

Deklarace :	**subroutine GdiLine (const pxy vector, var pxy origin)** **subroutine GdiLine (const pxy vector, const pxy origin)** **subroutine GdiLine (var pxy vector, var pxy origin)** **subroutine GdiLine (var pxy vector, const pxy origin)**
Parametr 1 :	**vector** odkaz na vektorové souřadnice definující úsečku
2 :	**origin** odkaz na souřadnice umístění grafického objektu (v pixelech)
Výstup :	- - -

GdiBeginPoly

GdiPoly

Podprogramy existují ve více variantách dle typu a umístění parametrů a jsou určeny pro iterativní kreslení víceúsekové čáry. Tento typ kreslení se dá použít třeba při vykreslování grafů nebo v případě, kdy body čáry nejsou k dispozici všechny naráz a jsou získávány postupně (anebo je to tak z různých důvodů programátorsky výhodnější).

Parametr **segment** (typu **linept**) udává souřadnici zlomového bodu a způsob jakým k němu bude vedena spojnice od předchozího zlomového bodu (viz kap. 5).

Podprogram GdiBeginPoly, kreslení zahajuje, v každém dalším kroku se pak přidává další bod čáry pomocí podprogramu GdiPoly.

Deklarace :	**subroutine GdiBeginPoly (var linept segment)** **subroutine GdiBeginPoly (const linept segment)** **subroutine GdiPoly (var linept segment)** **subroutine GdiPoly (const linept segment)**
Parametr :	**segment** odkaz na souřadnici a typ zlomového bodu
Výstup :	- - -

Pozn.: Podprogram GdiBeginPoly nemusíme použít v případě, že aktivně nastavujeme u struktury linept položku mode. Použití GdiBeginPoly je ale nutné v případě, že chceme iterativně nakreslit uzavřený mnohoúhelník, který v závěru vyplníme pomocí volání podprogramu GdiFillPoly

GdiFillPoly

Provede uzavření naposled iterativně kresleného mnohoúhelníku čárou vedenou z posledního nakresleného bodu do bodu nakresleného voláním podprogramu GdiBeginPoly. Následně najde vnitřní bod uzavřené oblasti a z tohoto bodu směrem do okolí provede vyplnění mnohoúhelníku barvou popředí (definovanou funkcí GdiFColor). Algoritmus pracuje správně jen pokud čára ohraničuje jedinou a uzavřenou oblast.

Deklarace :	**subroutine GdiFillPoly ( )**
Výstup :	- - -

Pozn.: GdiFillPoly provádí operaci nad naposled kresleným N-úhelníkem.

GdiNAngle

Podprogram existuje ve více variantách dle typu a umístění parametrů a kreslí n-úhelník / více úsekovou čáru z maximálně 256 segmentů.

Tabulka segmentů je zadána strukturovanou proměnnou **nangle** (blíže viz kap.5). Struktura je dimenzována na max. 256 segmentů.

Deklarace :	**subroutine GdiNAngle (const nangle tbl, var pxy origin)** **subroutine GdiNAngle (const nangle tbl, const pxy origin)** **subroutine GdiNAngle (var nangle tbl, var pxy origin)** **subroutine GdiNAngle (var nangle tbl, const pxy origin)**
Parametr 1 :	**tbl** odkaz na tabulku segmentů pro tvorbu N-úhelníku nebo lomené čáry
2 :	**origin** odkaz na souřadnice umístění grafického objektu (v pixelech)
Výstup :	- - -

GdiFillNAngle

Podprogram existuje ve více variantách dle typu a umístění parametrů a kreslí n-úhelník podobně jako GdiNAngle, s tím rozdílem že je vyplněný.

Aby bylo vyplnění n-úhelníku úspěšné musí být všechny body zadány s atributem **_lineto** (resp. s položkou mode=1). Po vykreslení hraniční čáry funkce uzavře prostor čarou z koncového do počátečního bodu. Dále je vyhledán vnitřní bod mnohoúhelníku a z tohoto bodu je mnohoúhelník vyplněn aktuální barvou pera. Vyplňovací algoritmus zklame v okamžiku, kdy se výsledný mnohoúhelník skládá z více než jedné oblasti nebo v případě, že hraniční čáry vrcholu svírají příliš ostrý úhel, který způsobí, že oblast neobsahuje kontinuální plochu nevyplněných bodů.

Deklarace :	**subroutine GdiFillNAngle (const nangle tbl, var pxy origin)** **subroutine GdiFillNAngle (const nangle tbl, const pxy origin)** **subroutine GdiFillNAngle (var nangle tbl, var pxy origin)** **subroutine GdiFillNAngle (var nangle tbl, const pxy origin)**
Parametr 1 :	**tbl** odkaz na tabulku segmentů pro tvorbu N-úhelníku nebo lomené čáry
2 :	**origin** odkaz na souřadnice umístění grafického objektu (v pixelech)
Výstup :	- - -

GdiRect

Existuje ve více variantách dle typu a umístění parametrů a vykreslí obdélník zadaný rozměrem **size** a umístěný na souřadnici **origin**.

Deklarace :	**subroutine GdiRect (const pxy size, var pxy origin)** **subroutine GdiRect (const pxy size, const pxy origin)** **subroutine GdiRect (var pxy size, var pxy origin)** **subroutine GdiRect (var pxy size, const pxy origin)**
Parametr 1 :	**size** odkaz na x-y rozměr obdélníku
2 :	**origin** odkaz na souřadnice umístění grafického objektu (v pixelech)
Výstup :	- - -

GdiFillRect

Existuje ve více variantách dle typu a umístění parametrů a nakreslí vyplněný obdélník zadaný rozměrem **size** a umístěný na souřadnici **origin**.

Deklarace :	**subroutine GdiFillRect (const pxy size, var pxy origin)** **subroutine GdiFillRect (const pxy size, const pxy origin)** **subroutine GdiFillRect (var pxy size, var pxy origin)** **subroutine GdiFillRect (var pxy size, const pxy origin)**
Parametr 1 :	**size** odkaz na x-y rozměr obdélníku
2 :	**origin** odkaz na souřadnice umístění grafického objektu (v pixelech)
Výstup :	- - -

GdiCirc

Existuje ve více variantách dle typu a umístění parametrů a kreslí elipsu/kružnici zadanou středovým bodem (**center**) a poloměrem v ose x a y (**radius**). Pokud jsou poloměry v ose x a y totožné, kreslí podprogram kružnici, opačném případě pak elipsu. Parametr **center** má význam obvyklého parametru **origin**, s tím rozdílem, že zde znamená střed objektu.

Deklarace :	**subroutine GdiCirc (const pxy center, var pxy radius)** **subroutine GdiCirc (const pxy center, const pxy radius)** **subroutine GdiCirc (var pxy center, var pxy radius)** **subroutine GdiCirc (var pxy center, const pxy radius)**
Parametr 1 :	**center** odkaz na souřadnici středu grafického objektu (v pixelech)
2 :	**radius** odkaz na velikost poloměrů v ose x a y (v pixelech)
Výstup :	- - -

GdiFillCirc

Existuje ve více variantách dle typu a umístění parametrů a funguje přesně stejně jako GdiCirc, s tím rozdílem, že kreslí vyplněnou kružnici/elipsu.

Deklarace :	**subroutine GdiFillCirc (const pxy center, var pxy radius)** **subroutine GdiFillCirc (const pxy center, const pxy radius)** **subroutine GdiFillCirc (var pxy center, var pxy radius)** **subroutine GdiFillCirc (var pxy center, const pxy radius)**
Parametr 1 :	**center** odkaz na souřadnici středu grafického objektu (v pixelech)
2 :	**radius** odkaz na velikost poloměrů v ose x a y (v pixelech)
Výstup :	- - -

GdiArc

Existuje ve více variantách dle typu a umístění parametrů a kreslí kruhový oblouk jehož parametry jsou zadány strukturovanou proměnnou **drw** typu arcmode. Struktura sdružuje tyto parametry: počáteční úhel, úhlová velikost výseče, poloměr.

Deklarace :	**subroutine GdiArc (const arcmode drw, var pxy origin)** **subroutine GdiArc (const arcmode drw, const pxy origin)** **subroutine GdiArc (var arcmode drw, var pxy origin)** **subroutine GdiArc (var arcmode drw, const pxy origin)**	
Parametr 1 :	**drw**	odkaz na datovou strukturu s parametry výseče
2 :	**origin**	odkaz na souřadnice umístění grafického objektu (v pixelech)
Výstup :	- - -	

Blíže k zadání parametrů výseče viz kapitola 5 - struktura **arcmode**. Kruhový oblouk má svůj virtuální střed umístěn na souřadnici dané parametrem **origin**. Oblouk je kreslen zadanou barvou a stopou pera (viz GdiPenType).

Pozn.: Struktura arcmode je totožná pro funkce GdiArc, GdiSlice a GdiFillSlice.

GdiSlice

Existuje ve více variantách dle typu a umístění parametrů a nakreslí obrys kruhové výseče (kruhový oblouk včetně krajních spojnic jeho koncových bodů do středu kružnice). Parametry jsou zadány strukturovanou proměnnou **drw** typu arcmode. Struktura sdružuje tyto parametry: počáteční úhel, úhlová velikost výseče, poloměr.

Deklarace :	**subroutine GdiSlice (const arcmode drw, var pxy origin)** **subroutine GdiSlice (const arcmode drw, const pxy origin)** **subroutine GdiSlice (var arcmode drw, var pxy origin)** **subroutine GdiSlice (var arcmode drw, const pxy origin)**	
Parametr 1 :	**drw**	odkaz na datovou strukturu s parametry výseče
2 :	**origin**	odkaz na souřadnice umístění grafického objektu (v pixelech)
Výstup :	- - -	

Blíže k zadání parametrů výseče viz kapitola 5 - struktura **arcmode**.

Kruhová výseč má svůj střed (vrcholový bod, kde se stýkají obě spojnice) umístěn na souřadnici dané parametrem **origin**. Výseč je kreslena zadanou barvou a stopou pera.

Pozn.: Struktura arcmode je totožná pro funkce GdiArc, GdiSlice a GdiFillSlice.

GdiFillSlice

Existuje ve více variantách dle typu a umístění parametrů a kreslí kruhovou výseč (kruhový oblouk včetně krajních spojnic jeho koncových bodů do středu kružnice), včetně výplně. Parametry jsou zadány strukturovanou proměnnou **drw** typu arcmode. Struktura sdružuje tyto parametry: počáteční úhel, úhlová velikost výseče, poloměr. Blíže k zadání parametrů výseče viz kapitola 5 - struktura **arcmode**. Kruhová výseč má svůj střed (vrcholový bod) umístěn na souřadnici dané parametrem **origin**. Výseč, včetně vnitřní výplně je kreslena zadanou barvou a stopou pera.

Deklarace :	**subroutine GdiFillSlice (const arcmode drw, var pxy origin)** **subroutine GdiFillSlice (const arcmode drw, const pxy origin)** **subroutine GdiFillSlice (var arcmode drw, var pxy origin)** **subroutine GdiFillSlice (var arcmode drw, const pxy origin)**	
Parametr 1 :	**drw**	odkaz na datovou strukturu s parametry výseče
2 :	**origin**	odkaz na souřadnice umístění grafického objektu (v pixelech)
Výstup :	- - -	

Pozn.: Struktura arcmode je totožná pro funkce GdiArc, GdiSlice a GdiFillSlice.

GdiBMP

Existuje ve více variantách dle typu a umístění parametrů a vykreslí rastrový obrázek (bitmapu) zadaný parametrem **bitmap**. Způsob vykreslení je řízen parametrem **control** (typ bmpmode). Význam a složení struktur viz kap. 5.

Deklarace :	**subroutine GdiBMP (const bmp bitmap, var bmpmode control)** **subroutine GdiBMP (const bmp bitmap, const bmpmode control)**	
Parametr 1 :	**bitmap**	odkaz na definici bitmapy v kódové (konstantní) paměti
2 :	**control**	odkaz na řídicí strukturu zobrazení bitmapy
Výstup :	- - -	

Pozn.: ve vrstvě GRAPHIC lze kreslit pouze s 256 barvami, tomu tedy musí odpovídat uložená bitmapa - nelze použít bitmapu s 65536 barvami.

Vrstva BACKGROUND

Vrstva BACKGROUND umožňuje pouze nastavení jednolité barvy pozadí, nebo vložení podkladového obrázku (bitmapy). Funkce které toto zajišťují jsou uvozeny písmeny "**GdiBkg**".

Vrstva BACKGROUND může pracovat 8-bitovými barvami (256 barev) i s 16-bitovými barvami (65536 barev).

GdiBkgSetColor

Nastavuje barvu podkladové plochy. Při předání parametru **color** typu byte je tento chápán jako index do palety 256 barev (8-bitová barva), při předání parametru **color** typu word je tento chápán jako 16-bitová barva v kódování RGB 5:6:5. Blíže k barvám viz stať BARVY.

Deklarace :	**subroutine GdiBkgSetColor (word color)** **subroutine GdiBkgSetColor (byte color)**
Parametr :	**color** zadání 8-bitové nebo 16-bitové barvy podkladu
Výstup :	- - -

Pozn.: Je-li nastaven podkladový obrázek (viz GdiBkgBMP), pak bude barva podkladové plochy zobrazována všude tam, kde bude mít bitmapa nastavenou průhlednou barvu.

GdiBkgBMP

Vykreslí ve vrstvě BACKGROUND rastrový obrázek (bitmapu) zadaný parametrem **bitmap**. Způsob vykreslení se zde na rozdíl od funkce GdiBMP (pro vrstvu GRAPHIC) nijak nenastavuje. Obrázek nelze otáčet, ani nijak umísťovat, musí být vždy na celý rozměr obrazovky.

Deklarace :	**subroutine GdiBMP (const bmp bitmap)**
Parametr 1 :	**bitmap** odkaz na definici bitmapy v kódové (konstantní) paměti
Výstup :	- - -

Pozn.: ve vrstvě BACKGROUND je možné použít bitmapu v 256 i v 65536 barvách.

Deklarace :	**subroutine GdiBkgBMP ( )**
Výstup :	- - -

Varianta GdiBMP bez parametru provede odstranění podkladového obrázku a jeho nahrazení jednolitou podkladovou barvou (buď výchozí nebo naposled zadefinovanou).

Komunikace přes linku RS485

PLC řady 400 umožňuje ovládat komunikaci na linkách RS485 z uživatelského programu. Funkce zajišťující toto ovládání jsou uvozeny písmeny "**Uart**".

UartConfig

Procedura nastaví parametry komunikace na vybrané lince podle aktuálního stavu položek *.cfg* a *.bdrate* strukturované proměnné **ua**. Výběr linky je přímo proveden hodnotou položky *.cfg*. Bezchybný průběh je indikován nastavením položky *.status* proměnné **ua** na hodnotu _ust_userstop. Část hodnoty položky *.cfg* představující číslo linky je třeba ponechat od prvního volání procedury UartConfig v programu bez změny.

Deklarace :	**subroutine UartConfig (var _uart ua)**
Parametr 1 :	**ua** proměnná se strukturou _uart pro reprezentaci jedné z linek RS485 v programu předaná s požadovaným nastavením konfiguračních parametrů
Výstup :	- - -

```
; Kód pro nastavení parametrů linky po resetu automatu.
var _uart ua      ; založení proměnné struktury _uart
if (RESET) then
     begin
     ua.cfg=0x06         ; USART0 (linka L1), 9 bitů, sudá parita, 1 stopbit
     ua.bdrate=57600     ; rychlost 57600 Bd
     UartConfig(ua)      ; nastavit parametry
     end
```

UartTxR

Procedura spustí novou transakci na lince. Proběhne v pořádku pouze, pokud již dříve proběhlo nastavení parametrů komunikace předáním té samé proměnné **ua** proceduře UartConfig a na lince žádná transakce neprobíhá. Nastavení nové transakce udávají položky *.tx_len*, *.rx_len*, *.rx_timeout1*, *.rx_timeout2* a *.d9b_mode* proměnné **ua**. Hodnota *d9b_mode* se uplatňuje pouze při komunikaci s 9 datovými bity na znak bez parity. Bezchybný průběh je indikován nastavením položky *.status* proměnné **ua** na hodnotu _ust_running (pokud předané parametry transakce odpovídají zahájení vysílání nebo příjmu), nebo _ust_done.

Deklarace :	**subroutine UartTxR (var _uart ua)**
Parametr 1 :	**ua** proměnná se strukturou _uart reprezentující linku RS485 v programu předaná s požadovaným nastavením parametrů transakce
Výstup :	- - -

```
; Kód pro zahájení vysílání na linku. Odvysílají se 2 znaky, pak se přepne
; na příjem a bude se maximálně 10 ms čekat na odpovědní rámec. Konec
; odpovědního rámce bude vyhodnocen při pauze mezi příchozími znaky větší,
; než odpovídá době trvání 3/2 znaku při rychlosti komunikace 9600 Bd a 11
; bitech na znak.
ua.tx_buf[0]=0x0C       ; 1. znak příkazu
```

```
ua.tx_buf[1]=0x24       ; 2. znak příkazu
ua.tx_len=2             ; příkaz tvoří 2 znaky
ua.rx_len=65535         ; nevíme přesně, kolik znaků očekávat jako odpověď
ua.rx_timeout1=100      ; 10*10 (jednotka je 1/10 ms)
ua.rx_timeout2=18       ; 10* 3/2 * 11/9600 * 1000 (jednotka je 1/10 ms)
UartTxR(ua)             ; zahájení transakce
```

UartStopTxR

Procedura zastaví případnou probíhající transakci na lince. Proběhne v pořádku pouze, pokud již dříve proběhlo nastavení parametrů komunikace předáním té samé proměnné **ua** proceduře UartConfig. Bezchybný průběh je indikován nastavením položky *.status* proměnné **ua** na hodnotu _ust_userstop.

Deklarace :	**subroutine UartStopTxR (var _uart ua)**
Parametr 1 :	**ua** proměnná se strukturou _uart reprezentující linku RS485 v programu
Výstup :	- - -

CRC16_RS

CRC16_LS

V programu pro PLC řady 400 lze snadno počítat kontrolní součet CRC-16 nad datovými bajty. Primárně je uvažováno využití výpočtu při implementaci komunikačních protokolů linky RS485. Funkce vrací výsledek výpočtu CRC-16 z prvních **len** bajtů v poli **b** typu _uart_buffer (pole velikosti 512 bajtů). Parametr **len** je tak třeba zadat v rozsahu 1 až 512. Datová struktura _uart_crc obsahuje dvě položky typu word, *.poly* a *.crc*. Proměnnou **p** se tak předávají dva parametry výpočtu, dělící polynom a inicializační hodnota výpočtu. Např. protokol Modbus RTU používá hodnotu polynomu 0xA001 a inicializační hodnotu 0xFFFF.

Deklarace :	**function word CRC16_RS(var _uart_buffer b, word len, var _uart_crc p)** **function word CRC16_RS(var _uart_buffer b, word len, const _uart_crc p)** **function word CRC16_LS(var _uart_buffer b, word len, var _uart_crc p)** **function word CRC16_LS(var _uart_buffer b, word len, const _uart_crc p)**
Parametr 1 :	**b** pole 512 bajtů typu _uart_buffer s daty pro výpočet
2 :	**len** počet bajtů v poli **b** určených pro výpočet
3 :	**p** parametry výpočtu
Výstup :	**crc** výsledek výpočtu

Jsou implementovány dvě varianty výpočtu. Variantě CRC16_RS odpovídá výpočet začínající od nejnižšího bitu (LSB) každého datového bajtu, variantě CRC16_LS naopak výpočet od nejvyššího bitu (MSB). U protokolů na rozhraní USART se dá předpokládat použití algoritmu CRC16_RS, protože datové bity po lince přicházejí postupně od LSB.

Využití funkcí je uvažováno především při práci s linkou RS485 v programu za pomoci dostupných prostředků využívajících ovladač USART. Lze proto velmi snadno počítat CRC-16 nad daty uloženými v polích *.tx_buf* a *.rx_buf* proměnné se strukturou _uart.

```
var _uart ua      ; založení proměnné struktury _uart
code _uart_crc crcpar = (0xA001,0xFFFF)  ; konstantní parametry výpočtu CRC
var word crc      ; proměnná pro uložení výsledku výpočtu CRC
......
crc = CRC16_RS(ua.rx_buf,ua.rx_n,crcpar) ; počítáme CRC z dat přijatých
                                         ; ovladačem USART
......
```

Zpracování textu a dat

Protože v programu psaném v jazyce SIMPLE 4 nejsou dovoleny programové cykly, je například inicializace proměnných programu do výchozího stavu (tovární nastavení) poměrně komplikovaná a na pozornost náročná úloha. Proto jsou k dispozici systémové funkce, které danou úlochu značně zjednodušují.

Copy

Funkce pro kopírování bloků dat, datových struktur.

Deklarace :	**function int Copy (var dataptr dst, var dataptr src)** **function int Copy (var dataptr dst, const dataptr src)** **function int Copy (var dataptr dst, dataptr src)** **function int Copy (dataptr dst, var dataptr src)** **function int Copy (dataptr dst, const dataptr src)** **function int Copy (dataptr dst, dataptr src)**	
Parametr 1 :	**dst**	odkaz na cíl kam budou kopírovány data
2 :	**src**	odkaz na zdroj dat pro kopírování
Výstup :	**len**	počet zkopírovaných bajtů paměti, 0 → data nelze kopírovat (chyba)

Indexované varianty

Kopírovací funkce je doplněna o počáteční indexy zdrojové a cílové paměťové lokace. Indexy jsou chápány v bajtech. Pokud chceme kopírovat od položky 5 pole wordů musíme index vypočítat tak, že číslo položky vynásobíme její délkou tj. hodnotou 2. Výsledný index pak bude mít hodnotu 10.

```
code word[10] firemni = (0,1,2,3,4,5,6,7,8,9)
Copy(@stackw,20 * 2,@firemni,0) ; kopíruje data z pole firemní do
; zásobníku automatu od položky 20
```

Příklad ukazuje kopírování celého pole firemní na zásobník od položky stackw[20] po položku stackw[29], kam bude uložena poslední hodnota 9 z pole firemní.

Deklarace :	**function int Copy (var dataptr dst, longword didx, var dataptr src, longword sidx)** **function int Copy (var dataptr dst, longword didx, const dataptr src, longword sidx)** **function int Copy (var dataptr dst, longword didx, dataptr src, longword sidx)** **function int Copy (dataptr dst, longword didx, var dataptr src, longword sidx)** **function int Copy (dataptr dst, longword didx, const dataptr src, longword sidx)** **function int Copy (dataptr dst, longword didx, dataptr src, longword sidx)**
Parametr 1 :	**dst** odkaz na cíl kam budou kopírovány data
2 :	**didx** index v cílovém prostoru od něhož budou data uložena
3 :	**src** odkaz na zdroj dat pro kopírování
4 :	**sidx** index ve zdrojovém prostoru od něhož budou data kopírována
Výstup :	**len** počet zkopírovaných bajtů paměti, 0 → data nelze kopírovat (chyba)

Indexované varianty s délkou zdroje

Deklarace :	**function int Copy (var dataptr dst, longword dstidx, var dataptr src, longword srcidx, longword srclen)** **function int Copy (var dataptr dst, longword dstidx, const dataptr src, longword srcidx, longword srclen)** **function int Copy (var dataptr dst, longword dstidx, dataptr src, longword srcidx, longword srclen)** **function int Copy (dataptr dst, longword dstidx, var dataptr src, longword srcidx, longword srclen)** **function int Copy (dataptr dst, longword dstidx, const dataptr src, longword srcidx, longword srclen)** **function int Copy (dataptr dst, longword dstidx, dataptr src, longword srcidx, longword srclen)**
Parametr 1 :	**dst** odkaz na cíl kam budou kopírovány data
2 :	**dstidx** index v cílovém prostoru od něhož budou data uložena
3 :	**src** odkaz na zdroj dat pro kopírování
4 :	**srcidx** index ve zdrojovém prostoru od něhož budou data kopírována
5 :	**srclen** počet byte, který bude kopírován
Výstup :	**len** počet zkopírovaných bajtů paměti, 0 → data nelze kopírovat (chyba)

U těchto variant krom specifikace zdroje, cíle, počátečních indexů zadáváme i počet kopírovaných byte v parametru srclen. Výsledný počet kopírovaných byte je možné popsat jako

Počet Byte = min(sizeof(src), srcidx + srclen)

```
code word[10] firemni = (0,1,2,3,4,5,6,7,8,9)
Copy(@stackw,20 * 2,@firemni,0,4) ; kopíruje data z pole firemní do
; zásobníku automatu od položky 20, délka 4 / 2 = 2 wordy
```

Uvedený příklad tedy kopíruje z pole firemní hodnoty 0 a 1 do zásobníku stackw[20] a stackw[21]

Speciální varianty pro typ string

Týkají se kopírování textů a to z kódové paměti (konstrukce "code" nebo "table") . Protože se speciálně u typu string musí používat firmwarové prostředky pro výpočet celkového počtu byte, mají tyto speciální funkce použity jako vstupní parametry zdroje dat přímo datové typy string

Deklarace :	**function int Copy (dataptr dst, const string src)** **function int Copy (var dataptr dst, const string src)** **function int Copy (dataptr dst, const string src, longword srcidx)** **function int Copy (var dataptr dst, const string src, longword srcidx)** **function int Copy (dataptr dst, const string src, longword srcidx, longword srclen)** **function int Copy (var dataptr dst, const string src, longword srcidx, longword srclen)**
Parametr 1 :	**dst** odkaz na cíl kam budou kopírovány data
2 :	**src** odkaz na zdroj dat pro kopírování
3 :	**dstidx** index v cílovém prostoru od něhož budou data kopírována
4 :	**srcidx** index znaku stringu od něhož budou data kopírována
5 :	**srclen** počet znaků stringu které se budou kopírovat
Výstup :	**len** počet zkopírovaných bajtů paměti, 0 → data nelze kopírovat (chyba)

Vzhledem k tomu, že v jazyce Simple 4 je pro ukončení textového řetězce použito hodnoty 0, kopírují všechny funkce text včetně ukončující 0, za předpokladu dostatku místa v cílové struktuře. Pokud ne kopíruje se maximum položek. Počet kopírovaných byte, který funkce vrací, je počet zapsaných znaků řetězce tj. bez započítání koncové hodnoty 0. Pokud se kopíruje pouze část textového řetězce, pak se kopírování zakončí hodnotou 0. Pokud část řetězce daná parametrem srclen přesáhne délku řetězce, zkopíruje se řetězec do konce bez ohledu na hodnotu tohoto parametru.

Fill

Slouží k blokové inicializaci paměti zadanou hodnotou.

Deklarace :	**function int Fill (var dataptr dst, byte val)** **function int Fill (dataptr dst, byte val)**	
Parametr 1 :	**dst**	odkaz na cíl kde budou ukládány hodnoty **val**
2 :	**val**	inicializační hodnota
Výstup :	**len**	počet zkopírovaných bajtů paměti, 0 → data nelze kopírovat (chyba)

StrLen

Funkce je určena pro zjišťování délky řetězce znaků (textu).

Deklarace :	**function int StrLen (var dstring txt)** **function int StrLen (const string txt)**	
Parametr 1 :	**txt**	řetězec textu, jehož délku zjišťujeme
Výstup :	**len**	délka zadaného textu (řetězce)

```
code string text = "Toto je text"
var dstring dst
var int len
Copy(@dst,text)
len = StrLen(dst)
```

StrIns

Umožňuje vložení znaků textu od zadané pozice v cílovém řetězci.

Deklarace :	**function int StrIns (var dstring dst, longword dstidx, byte char, longword num, longword maxlen)**	
Parametr 1 :	**dst**	cílový retězec textu
2 :	**dstidx**	index (pozice), kam bude vložen první znak
3 :	**char**	vkládaný znak
4 :	**num**	počet vkládaných znaků **char**
5 :	**maxlen**	maximální počet znaků výsledného řetězce
Výstup :	**len**	výsledná délka cílového řetězce <= maxlen

Znaky z cílového řetězce, jsou od zadané pozice připojeny za vkládaný text. Pokud není v cílovém řetězci na připojení místo, jsou přebytečné znaky odstraněny. Speciální systémová konstanta str2end označuje, že znaky vyplní řetězec od zadané pozice do konce.

```
var dstring text
var int len
len = StrIns(text,0,' ',str2end, str2full) ; str2full-využije maximální délku
```

Deklarace :	**function int StrIns (var dstring dst, longword dstidx, var dstring src, longword num, longword maxlen)** **function int StrIns (var dstring dst, longword dstidx, const string src, longword num, longword maxlen)**	
Parametr 1 :	**dst**	cílový retězec textu
2 :	**dstidx**	index (pozice), kam bude vložen první znak
3 :	**src**	vkládaný řetězec
4 :	**num**	počet vkládaných znaků z řetězce (**str2full** – vloží se celý řetězec)
5 :	**maxlen**	maximální počet znaků výsledného řetězce
Výstup :	**len**	výsledná délka cílového řetězce <= maxlen

```
code string text = "Toto je text"
code string take = "take"
var dstring dst
var int len
Copy(@dst,text)
len = StrIns(dst, 8, take, str2full, 12) ; "Toto je take" - zakrácení na 12
```

StrRep

Slouží k přepsání znaků textu od zadané pozice v cílovém řetězci znaky řetězce nového. Délka cílového řetězce se nemění.

Deklarace :	**function int StrRep (var dstring dst, longword dstidx, byte char, longword num, longword maxlen)**	
Parametr 1 :	**dst**	cílový retězec textu
2 :	**dstidx**	index (pozice), kam bude vložen první znak
3 :	**char**	znak, kterým nahrazujeme znaky řetězce
4 :	**num**	počet nahrazovaných znaků znakem **char**
5 :	**maxlen**	maximální počet znaků výsledného řetězce
Výstup :	**len**	výsledná délka cílového řetězce <= maxlen

```
var dstring text
StrRep(text,0,' ',str2end,13) ;nahradit a zakrátit na 13 znaků
```

Deklarace :	**function int StrRep (var dstring dst, longword dstidx, var dstring src, longword num, longword maxlen)** **function int StrRep (var dstring dst, longword dstidx, const string src, longword num, longword maxlen)**	
Parametr 1 :	**dst**	cílový řetězec textu
2 :	**dstidx**	index (pozice), kde bude nahrazen první znak
3 :	**src**	řetězec, kterým přepisujeme znaky cílového řetězce
4 :	**num**	počet nahrazovaných znaků (**str2end** – použije se celý řetězec)
5 :	**maxlen**	maximální počet znaků výsledného řetězce
Výstup :	**len**	výsledná délka cílového řetězce <= maxlen

```
code string text = "Toto je text"
code string tady = "Tady"
var dstring dst
Copy(@dst,text)
StrRep(dst, 0, tady,src2end,20) ; "Tady je text"
```

StrDel

Maže (zkracuje) část nebo celý řetězec znaků

Deklarace :	**function int StrDel (var dstring dst, longword dstidx, longword num, longword maxlen)**	
Parametr 1 :	**dst**	cílový řetězec textu
2 :	**dstidx**	index (pozice), od které bude řetězec mazán
3 :	**num**	počet znaků, které požadujeme smazat
4 :	**maxlen**	maximální počet znaků výsledného řetězce
Výstup :	**len**	nová délka cílového řetězce

```
code string text = "Toto je text"
var dstring dst
Copy(@dst,text)
StrDel(dst, 0, 5, 5) ; "je te" vymazat 5 znaků a zakrátit na 5 znaků
```

Datové přenosy

Automaty řady 400 mají řadu komunikačních možností přes vestavěné sériové komunikační rozhraní. Jsou k dispozici systémové funkce umožňující snadnou realizaci kontinuální komunikace přes obecné sériové rozhraní. Pomocí funkce pro iterativní zpracování textových dat, je možné realizovat libovolný textový komunikační protokol v obou směrech tj. jak pro vysílání, tak pro příjem.

Funkce pro zpracování přenosů dat pracují s datovou strukturou _circular_buffer, která představuje „nekonečné“ pole dat. Toto pole je při komunikaci zpracováváno postupně a s pomocí funkcí pro zjištění volného místa a počtu uložených dat, je možné přizpůsobit datový tok komunikačním možnostem sériového rozhraní.

TrResetBuffer

Slouží k inicializaci datové struktury kruhového bufferu do výchozího stavu.

Deklarace :	**subroutine TrResetBuffer (var _circular_buffer b)**
Parametr 1 :	**b** odkaz na datovou strukturu kruhového bufferu
Výstup :	**--**

```
var _circular_buffer buf
TrResetBuffer(buf) ; inicializuje datovou strukturu
```

TrWriteBuffer

Existuje v řadě variant podle datového typu, který je ukládán do struktury kruhového bufferu.

Deklarace :	**function word TrWriteBuffer (var _circular_buffer b, byte val)** **function word TrWriteBuffer (var _circular_buffer b, word val)** **function word TrWriteBuffer (var _circular_buffer b, int val)** **function word TrWriteBuffer (var _circular_buffer b, longword val)** **function word TrWriteBuffer (var _circular_buffer b, longint val)** **function word TrWriteBuffer (var _circular_buffer b, float val)** **function word TrWriteBuffer (var _circular_buffer b, const string val)** **function word TrWriteBuffer (var _circular_buffer b, var dstring val)** **function word TrWriteBuffer (var _circular_buffer b, var _io_buf val)**
Parametr 1 :	**b** odkaz na datovou strukturu kruhového bufferu
2 :	**val** hodnota ukládaná do bufferu
Výstup :	**wlen** <> 0 - počet zapsaných byte, 0 – zápis neproběhl (plný buffer)

Varianta pro datový typ _uart_buffer

Varianta pro zpracování dat z datového typu _uart_buffer.

Deklarace :	**function word TrWriteBuffer (var _circular_buffer b, var _uart_buffer val, word len)**	
Parametr 1 :	**b**	odkaz na datovou strukturu kruhového bufferu
2 :	**val**	hodnota ukládaná do bufferu
3 :	**len**	počet platných byte v _uart_bufferu
Výstup :	**wlen**	<> 0 - počet zapsaných byte, 0 – zápis neproběhl (plný buffer)

Varianty pro datový typ dataptr

Funkce pro zápis do kruhového bufferu z datového typu dataptr se liší návratovou hodnotou, která je typu word a obsahuje počet zapsaných bajtů do kruhového bufferu.

Deklarace :	**function word TrWriteBuffer (var _circular_buffer b, const dataptr val)** **function word TrWriteBuffer (var _circular_buffer b, dataptr val)**	
Parametr 1 :	**b**	odkaz na datovou strukturu kruhového bufferu
2 :	**val**	hodnota ukládaná do bufferu
Výstup :	**wlen**	počet zapsaných byte

Varianta pro datový typ _io_buf

Funkce pro zápis do kruhového bufferu existuje ve speciálním tvaru pro datovou strukturu **_io_buf**, kterou používají komunikační uzly typu RD_BUFFER a WR_BUFFER. Varianta respektuje počet platných byte daném ve struktuře položkou **len** a kopíruje tak pouze platný obsah položky **data**.

Deklarace :	**function word TrWriteBuffer (var _circular_buffer b, var _io_buf val)**	
Parametr 1 :	**b**	odkaz na datovou strukturu kruhového bufferu
2 :	**val**	odkaz na strukturu **_io_buf**
Výstup :	**wlen**	počet zapsaných byte

Indexované varianty

Funkce pro zápis do kruhového bufferu existuje i v indexových variantách, pokud je zdrojem dat datový typ dataptr.

Deklarace :	**function word TrWriteBuffer (var _circular_buffer b, const dataptr val, longword idx)** **function word TrWriteBuffer (var _circular_buffer b, const dataptr val, longword idx, longword len)** **function word TrWriteBuffer (var _circular_buffer b, dataptr val, longword idx)** **function word TrWriteBuffer (var _circular_buffer b, dataptr val, longword idx , longword len)**	
Parametr 1 :	**b**	odkaz na datovou strukturu kruhového bufferu
2 :	**val**	hodnota ukládaná do bufferu
3 :	**idx**	index v paměti od něhož se začnou ukládat data do kruhového bufferu
4 :	**len**	počet zapisovaných byte
Výstup :	**wlen**	počet zapsaných byte

TrReadBuffer

Existuje v řadě variant podle datového typu, který je čten z kruhového bufferu.

Deklarace :	**function word TrReadBuffer (var _circular_buffer b, var byte val)** **function word TrReadBuffer (var _circular_buffer b, var word val)** **function word TrReadBuffer (var _circular_buffer b, var int val)** **function word TrReadBuffer (var _circular_buffer b, var longword val)** **function word TrReadBuffer (var _circular_buffer b, var longint val)** **function word TrReadBuffer (var _circular_buffer b, var float val)** **function word TrReadBuffer (var _circular_buffer b, var dstring val)** **function word TrReadBuffer (var _circular_buffer b, var _io_buf val)**	
Parametr 1 :	**b**	odkaz na datovou strukturu kruhového bufferu
2 :	**val**	hodnota vyčtená z kruhového bufferu
Výstup :	**rlen**	<> 0 počet přečtených byte, 0 – čtení neproběhlo (prázdný buffer)

Varianta pro datový typ _uart_buffer

Varianta pro zpracování dat z datového typu _uart_buffer.

Deklarace :	**function word TrReadBuffer (var _circular_buffer b, var _uart_buffer val, var word len)**	
Parametr 1 :	**b**	odkaz na datovou strukturu kruhového bufferu
2 :	**val**	odkaz na _uart_buffer kam budou vyčítána data
3 :	**len**	počet platných byte vyčtených do _uart_bufferu
Výstup :	**rlen**	<> 0 počet přečtených byte, 0 – čtení neproběhlo (prázdný buffer)

Varianty pro datový typ dataptr

Funkce pro čtení dat z kruhového bufferu do cílové paměti zadané pomocí datového typu dataptr se liší návratovou hodnotou, která je typu word a obsahuje počet přečtených bajtů do kruhového bufferu.

Deklarace :	**function word TrReadBuffer (var _circular_buffer b, dataptr val)**	
Parametr 1 :	**b**	odkaz na datovou strukturu kruhového bufferu
2 :	**val**	odkaz na cílovou paměť, kam jsou data vyčtena
Výstup :	**rlen**	počet vyčtených byte

Varianta pro datový typ _io_buf

Funkce pro čtení dat z kruhového bufferu existuje ve speciálním tvaru pro datovou strukturu **_io_buf**, kterou používají komunikační uzly typu RD_BUFFER a WR_BUFFER. Tato varianta respektuje maximální počet platných byte v položce data (tj. **32**). Při čtení dat z kruhového bufferu a jejich zápisu do struktury **_io_buf** tedy funkce nejprve přepíše max. 32 byte z kruhového bufferu do položky **data** struktury **_io_buf** a následně nastaví počet kopírovaných (platných) byte do položky **len.**

Deklarace :	**function word TrReadBuffer (var _circular_buffer b, var _io_buf val)**	
Parametr 1 :	**b**	odkaz na datovou strukturu kruhového bufferu
2 :	**val**	odkaz na strukturu **_io_buf**
Výstup :	**wlen**	počet zapsaných byte

Indexovaná varianta

Funkce pro čtení z kruhového bufferu existuje pro datový typ dataptr i v indexové variantě

Deklarace :	**function word TrReadBuffer (var _circular_buffer b, dataptr val, longword idx)**	
Parametr 1 :	**b**	odkaz na datovou strukturu kruhového bufferu
2 :	**val**	hodnota ukládaná do bufferu
3 :	**idx**	index v paměti od něhož se začnou vyčítat data z kruhového bufferu
Výstup :	**rlen**	počet vyčtených byte

TrGetBufferGap

Zjišťuje volné místo pro data v kruhovém bufferu.

Deklarace :	**function word TrGetBufferGap (var _circular_buffer b)**	
Parametr 1 :	**b**	odkaz na datovou strukturu kruhového bufferu
Výstup :	**num**	počet volných položek (bajtů) v kruhovém bufferu

TrGetBufferDataLen

Zjišťuje počet byte uložených v kruhovém bufferu.

Deklarace :	**function word TrGetBufferDataLen (var _circular_buffer b)**	
Parametr 1 :	**b**	odkaz na datovou strukturu kruhového bufferu
Výstup :	**num**	počet položek (bajtů) uložených v kruhovém bufferu

TrCompare

Pomocná funkce pro porovnání textových řetězců existuje v několika variantách podle typu předávaných parametrů. Funkce vrací rozdíl hodnot prvních nalezených odlišných znaků při porovnávání řetězců.

Deklarace :	**function word TrCompare (var dstring dst, var dstring src)** **function word TrCompare (var dstring dst, const string src)** **function word TrCompare (const string dst, var dstring src)**	
Parametr 1 :	**b**	odkaz na datovou strukturu kruhového bufferu
Výstup :	**dif**	0 = řetězce jsou totožné, <0 dst < src, >0 dst > src

TrSetParam

Funkce slouží k nastavení parametrů použitých pro zpracování dat komunikačními funkcemi systému.

Deklarace :	**subroutine TrSetParam (longword param)**
Parametr 1 :	**param** hodnota parametrů pro globální nastavení zpracování dat
Výstup :	**--**

param - hodnota určuje způsob zpracování dat.

bit 0 až 2 ... nastavuje typ zpracování čísel funkcí TrScanBuffer

- 0 … **_tr_number_auto** – automatické rozpoznání čísel hexadecimální zápis čísla se předpokládá ve formátu 0x..
- 1 … **_tr_number_sufix** – čísla jsou předpokládána pouze celá, přičemž se předpokládá rozlišení soustavy pomocí sufixu tj. d nebo D pro dekadické číslo, b nebo B pro binární, o nebo O pro oktalovou, h nebo H pro hexadecimální
- 2 … **_tr_number_hex** – čísla jsou zapsána v hexadecimální soustavě bez prefixu nebo sufixu
- 3 … **_tr_number_float** - předpokládá stejnou funkci jako _tr_number_auto s tím, že všechna čísla jsou ukládána v datovém typu float
- 4 … **_tr_number_none** – čísla nejsou funkcí TrScanBuffer detekována
- 5 … 7 hodnoty nejsou použity

bit 3 ... není použit

bit 4 až 5 ... volí styl zpracování textu a symbolů funkcí TrScanBuffer

- 0 … **_tr_auto_symbol** – za symbol jsou považovány všechny řetězce složené z písmen a současně je implementován převod na velká písmena pro kódování CE
- 1 … **_tr_text_symbol** – vše je považováno za text, kromě oddělovačů CR (kód 13) a LF (kód 10)
- 2 … **_tr_alfanumeric_symbol** – za symbol se považují všechny řetězce počínající písmenem a obsahující písmena a číslice

bit 6 ... **_tr_bigendian** – hodnota 1, volí ukládání vícebajtových datových typů ve formátu nejdříve nejvyšší byte a naposled nejnižší, hodnota 0 volí pořadí opačné tj. nejdříve nejnižší a nakonec nejvyšší

bit 7 ... **_tr_case_sensitive** – hodnota 1 volí citlivost funkce TrScanBuffer a TrCompare na malá a velká písmena, hodnota 0 vypíná citlivost na velká a malá písmena atp.

bity 8 až 31 ... jsou nevyužity

TrScanBuffer

Funkce slouží k iterativnímu zpracování textových dat z kruhového bufferu.

Deklarace :	**function bit TrScanBuffer (var _circular_buffer b, var _scan_value val)**
Parametr 1 :	**b** odkaz na datovou strukturu kruhového bufferu
2 :	**val** odkaz datovou strukturu **_scan_value** s aktuálně detekovaným obsahem
Výstup :	**ok** 1 = funkce detekovala obsah, 0 = funkce nezjistila nebo nemohla detekovat obsah

Použití funkce se předpokládá iterativním způsobem tj. přenášený textový obsah se zpracovává postupně po jednotlivých položkách. Každá detekovaná položka patří jedné ze čtyř skupin.

prázdný znak (white char) – označuje znak, který je při zpracování ignorován a funkce při nalezení takového znaku znak přeskočí a pokračuje ve zpracování znakem následujícím. Prázdný znak funguje též jako oddělovač/ukončovač při zpracování symbolů a čísel

významný znak (sign char) – skupina znaků, které jsou funkcí detekovány jednotlivě a mají nějaký význam pro další zpracování v uživatelském kódu. V případě analýzy aritmetické operace představují významný znak znaky „+“,„-“, „*“ a „/“. Významný znak slouží též jako oddělovač/ukončovač při zpracování symbolů a čísel

číslo – je představováno skupinou čísel a znaků vyjadřující zápis číselné hodnoty. Formát tohoto zápisu je dán aktuálním nastavením parametrů zpracování viz funkce TrSetParam

symbol – skupina znaků detekovaných společně jako klíčové slovo (text). Znaky, které jsou považovány za platné pro symbol určuje aktuální nastavení parametrů zpracování viz funkce TrSetParam

Krom prázdných znaků jsou všechny ostatní skupiny iterativně předávány do aplikace pomocí položek struktury **val** typu **_scan_value**.

Příklady zpracování textu

Pro ukázku použití funkce předpokládejme obsah kruhového bufferu ve tvaru

„d32 = 126\013“. Tento tvar představuje příkaz pro nastavení proměnné d32 na hodnotu 126. Zápis \013 v řetězci představuje oddělovač CR zapsaný číselným kódem v syntaxi jazyka SIMPLE 4.

```
; nastavení běžného zpracování symbolů
TrSetParam(0)          ; nastavení parametrů zpracování
TrScanBuffer(b,val)    ; val.s -> „d“, val.valid -> symbol
TrScanBuffer(b,val)    ; val.lw -> 32, val.valid -> číslo
TrScanBuffer(b,val)    ; val.lw = ‘=’, val.valid -> sign char
TrScanBuffer(b,val)    ; val.lw -> 126, val.valid -> číslo
TrScanBuffer(b,val)    ; val.lw = 13, val.valid -> sign char

; nastavení zpracování symbolů alfanumerických
```

```
TrSetParam(0x20)        ; nastavení parametrů zpracování
TrScanBuffer(b,val)     ; val.s -> „d32“, val.valid -> symbol
TrScanBuffer(b,val)     ; val.lw = ‘=’, val.valid -> sign char
TrScanBuffer(b,val)     ; val.lw -> 126, val.valid -> číslo
TrScanBuffer(b,val)     ; val.lw = 13, val.valid -> sign char

; nastavení zpracování prostého textu
TrSetParam(0x10)        ; nastavení parametrů zpracování
TrScanBuffer(b,val)     ; val.s -> „d32 = 126“, val.valid -> symbol
TrScanBuffer(b,val)     ; val.lw = 13, val.valid -> sign char
```

TrInitSymbol

Funkce umožňuje nastavit specifické parametry pro zpracování řetězců funkcí TrScanBuffer. Typickým příkladem použití může být řešení zpracování symbolů v režimu, kdy není zapnuta citlivost na velká a malá písmena a funkce TrScanBuffer vrací symbol ve tvaru velkých písmen. Systém ve výchozím nastavení zpracovává znakovou sadu CE a to se nemusí krýt s požadavkem uživatelského programu. Stejně tak je možné funkcí TrInitSymbol zajistit zpracování symbolů s převodem na všechna písmena malá.

Deklarace :	**subroutine TrInitSymbol (const _scan_symbol b)** **subroutine TrInitSymbol ()**
Parametr 1 :	**b** odkaz na seznam indexů písmen a jejich zpracování
Výstup :	**--**

Parametr _scan_symbol a jeho funkce

Parametr b (_scan_symbol) představuje pole 256 byte dlouhé. Každý byte tohoto pole má pozici v poli (index), který je systémem chápán jako kód písmena. Hodnota byte pak představuje návod, co z daným písmenem udělat . Každý byte v poli může mít hodnotu 0, 1 a nebo libovolnou hodnotu z rozsahu 32 - 256. Hodnoty v rozsahu 2-31 jsou rezervovány a v současnosti je funkce TrScanBuffer považuje za nevýznamné znaky. Následující tabulka ukazuje nastavení hodnot pro některé vybrané znaky v kódování CE (tabulka odpovídající celému poli hodnot by měla 256 řádků).

Nastavení podmínek pro zpracování textů je možné libovolně měnit, přičemž nezáleží na tom, v jaké fázi rozpracovanosti je právě vyhodnocovaný text. Varianta funkce TrInitSymbol bez parametrů slouží k inicializaci vyhodnocování podle naposledy použitého nastavení ve volání podprogramu TrSetParam. Pokud tento program nebyl nikdy volán, navrátí se vyhodnocování do výchozího stavu po resetu automatu tj. číslice se vyhodnocují podle automatických pravidel, symboly mohou obsahovat pouze písmena, převod na velká písmena je vypnutý a data jsou ukládána do kruhového bufferu ve formátu „little endian“.

Index (Znak)	Hodnota	Převod na velká písmena	Akce
32 (mezera)	0	-------	není zpracován - vynechá se
43 (+)	1	-------	významný znak - není součástí symbolu
48 (0)	48	48 (0)	číslice 0 nemá velkou a malou alternativu a může být součástí symbolu
65 (A)	65	65 (A)	velké písmeno A je součástí symbolu a při převodu na velká písmena je použito přímo
97(a)	65	65 (A)	malé písmeno a je součástí symbolu a při převodu na velké písmeno se použije hodnota 65 představující velké A
154(š)	138	138 (Š)	malé písmeno š je součástí symbolů a je při převodu na velké písmeno nahrazeno velkým Š. Pokud bychom použili hodnotu 83, pak bychom současně s převodem odstranili i háček a malé š bychom nahradili velkým S

TrFindDataRead

Funkce slouží k vyhledání sekvence znaků v kruhovém bufferu. Hodí se zejména ke hledání začátku či hlavičky při zpracování obecných dat komunikace. Funkce, které předáváme odkaz na kruhový buffer, hledaný vzorek dat ve formě string, dstring nebo dptr, má návratovou hodnotu typu bit. Pokud je vrácena hodnota 0, pak funkce vzorek nenalezla. V případě, že funkce vrátí 1, pak je vzorek nalezen a je možné ho přečíst pomocí vhodné varianty funkce TrReadBuffer. Funkce zahájí hledání vzorku tehdy, pokud je v kruhovém bufferu dostatek dat, tj. pokud kruhový buffer obsahuje větší nebo stejný počet byte než je délka zadaného vzorku (např. u varianty string se jedná o počet znaků). V případě, že kruhový buffer obsahuje méně byte než je délka vzorku, vrací funkce hodnotu 0 a žádné znaky nezpracovává. Funkce se tedy při zpracování posouvá po datech a současně uvolňuje kruhový buffer tak, aby vždy zbyl alespoň počet byte o jednu menší než je délka vzorku. V tomto případě již nelze vzorek porovnávat.

Deklarace :	**function bit TrFindDataRead (var _circular_buffer b, var dstring ptrn)** **function bit TrFindDataRead (var _circular_buffer b, const string ptrn)** **function bit TrFindDataRead (var _circular_buffer b, dataptr ptrn)**
Parametr 1 :	**b** odkaz na datovou strukturu kruhového bufferu
Parametr 2 :	**ptrn** hledaný vzorek dat
Výstup :	0 = vzorek nenalezen, 1 vorek nalezen a připraven ke čtení

Varianta funkce se zástupným znakem

Funkce je variantou základní funkce TrFindDataRead. Vůči předloze požaduje volání funkce o jeden parametr typu byte označený symbolem msk navíc. Tímto parametrem sdělujeme vyhledávací funkci, které hodnoty (znaky) ve vzorku dat nejsou podstatné co do velikosti ale jsou podstatné co do přítomnosti. Mějme například hlavičku komunikace xxxMICROPELyy, kde xxx a yy představují číselné hodnoty, které se mohou měnit např. 123MICROPEL45. Při zpracování komunikace, je ale nutné tyto údaje znát a současně je třeba nalézt řetězec MICROPEL. Z uvedených požadavků je zřejmé, že pokud chceme po úspěšném vyhledání zjistit hlavičku včetně proměnných znaků, musíme zařídit, aby je vyhledávací funkce nebrala v potaz hodnotou ale pouze jejich přítomností. Na to se právě použije parametr maska. Vyhledávaný řetězec předáme např. ve tvaru "???MICROPEL??" a do parametru msk předáme znak '?'. Vyhledávací funkce nyní zafunguje tak, že vyhledá řetezec MICROPEL a vrátí 1 (nalezeno), pokud za řetězcem MICROPEL budou v kruhovém bufferu ještě alespoň 2 znaky a pokud před tímto řetězcem byly v kruhovém bufferu minimálně 3 znaky. Ukazatel pro čtení kruhového bufferu bude nastaven na počátek nalezeného vzorku tj. na uvozující tři znaky na jejichž hodnotě nezáleží. Parametr msk tak označuje znak či hodnotu, která je ve vzorku zpracována pouze tím, že je přítomna.

Deklarace :	**function bit TrFindDataRead(var _circular_buffer b, var dstring ptrn, byte msk)** **function bit TrFindDataRead(var _circular_buffer b,const string ptrn, byte msk)** **function bit TrFindDataRead(var _circular_buffer b, dataptr ptrn, byte msk)**
Parametr 1 :	**b** odkaz na datovou strukturu kruhového bufferu
Parametr 2 :	**ptrn** hledaný vzorek dat
Parametr 3:	**msk** hodnota (znak), který je brán polohou, nikoli hodnotou
Výstup :	0 = vzorek nenalezen, 1 vorek nalezen a připraven ke čtení

5. DATOVÉ STRUKTURY

V konfiguraci překladače SIMPLE4 pro řadu 400 je již v základu předdefinovaná řada strukturovaných datových typů. Je to hlavně kvůli novým funkcím SIMPLE4 pro tuto řadu PLC, které počítají s předáváním složených proměnných s určitou konkrétní strukturou. Při využívání těchto funkcí v uživatelském programu je tedy třeba nejprve založit proměnnou daného typu a pak ji předat do vestavěné funkce SIMPLE4, která nad ní provede požadovanou operaci.

Příklad:

```
; vyčtení informací o zařízení EXbus na adrese 1 (vnitřní I/O modul na střední
pozici)
;-------------------------------------------------------------------------
  var  _io_dsc  ModulInfo     ; založení příslušné strukturované proměnné
  GetIONodeDef(1, ModulInfo)  ; předání struktury odkazem
```

rtc_type

Struktura obsahující položky přesně odpovídající registrům reálného času, definovaným shodně na všech PLC MICROPEL. S použitím této struktury lze unifikovat funkce pracující s reálným časem tak, aby operovaly ve stejném formátu jak s obvodem RTC v automatu, tak s položkami v paměti (editace času/data v kalendářích, databázích atd.). Je předdefinována i nová proměnná SYS_RTCDevice, která je právě typu **rtc_type**. Původní registry RTC, tak jak jsou na PLC MICROPEL od počátku, zůstávají k dispozici i nadále. Lze tedy na RTC přistupovat dvojím způsobem.

Použití :	**pro operace s obvodem reálného času/kalendáře (RTC)** **pro uživatelské operace s proměnnými data/času typu rtc_type**
Definice :	type struct word SECOND, ; sekundy word MINUTE, ; minuty word HOUR, ; hodiny word DAY, ; datum - den word MONTH, ; datum - měsíc word YEAR, ; datum - rok word WEEK ; den v týdnu (1=Ne, 2=Po ..... 7=So) end **rtc_type**

Příklad:

```
; POROVNÁNÍ REGISTRU HODIN:
;-----------------------------------------
if HOUR=8 then Zap=1                ; klasický přístup přes registr HOUR
if SYS_RTCDevice.HOUR=8 then Zap=1  ; přístup přes strukturovanou proměnnou
```

Příklad:

Funkce MeDispRTC v knihovně MenuLIB zobrazuje v různých formátech reálný čas na displej a umí pracovat i se strukturou **rtc_type**. Lze ji tedy použít jednak pro zobrazení aktuálního času/data z RTC, jednak pro zobrazení např. uložených časových značek v archivu událostí:

```
;--------------------- definice struktury archivního záznamu --------------
type struct                          ; definice struktury archivního záznamu
  byte      udalost,                 ; číslo události
  word      parametr,                ; parametr k události
  rtc_type  cas                      ; čas složený z položek struktury rtc_type
end  archivnizaznam
;--------------------- PROGRAM ----------------------------------------------
var archivnizaznam Archiv[256]       ; založení pole archivních záznamů
......
MeDispRTC(0x1F, SYS_RTCDevice)       ; zobrazení aktuálního reálného času
MeDispRTC(0x1F, Archiv[index].cas) ; zobrazení času z archivního záznamu
......
```

pxy

Má položky **x** a **y**. Využitelná všude, kde se pracuje s 2D pravoúhlými souřadnicemi.

Díky tomu, že položky jsou typu int (tedy znaménkové), lze strukturu využít i pro zadávání záporných diferencí a posunů - tedy definovat i vektorové veličiny v celém 2D prostoru.

Použití :	**v různých podpůrných zobrazovacích funkcích** **v kreslicích funkcích Gdi.. pro vrstvu GRAPHIC** **obecně všude, kde se pracuje s 2D souřadnicemi x-y**
Definice :	type struct int x, ; souřadnice x int y ; souřadnice y end **pxy**

V grafických a zobrazovacích funkcích se používá hned v několika významech:

size - pxy zde má význam xy rozměru objektu (např. v pixelech nebo ve znacích)

org - pxy zde má význam xy souřadnice umístění objektu (v pixelech)

tl, br - pxy zde má význam xy souřadnice levého-horního / pravého-dolního rohu (v pixelech)

vector - pxy zde má význam xy diference (v pixelech)

bmp

Detailní znalost této struktury není při tvorbě programů nutná. Struktura je většinou generována podpůrnými nástroji pro začlenění rastrových obrázků (bitmap) do programu. Je využitelná pouze pro konstanty, nelze ji použít např. pro proměnné vytvořené uživatelem v datové paměti. Strukturovaná konstanta typu bmp, která vznikne při začleňování bitmapy podpůrnými nástroji do uživatelského programu, je de-facto součástí vygenerovaného kódu bitmapy. Její název reprezentuje danou bitmapu v uživatelském programu a zadává se coby reference na bitmapu do grafických funkcí pro kreslení bitmap.

Použití :	**ve funkcích pracujících s bitmapami**
Definice :	type struct word width, ; šířka bitmapy (v pixelech) word height, ; výška bitmapy (v pixelech) word slides, ; počet snímků word mode, ; barevný mód (8/16 bitů) string ptr ; odkaz na data bitmapy end **bmp**

bmpmode

Proměnné s touto strukturou vytváří uživatel v datové paměti a předává je do funkcí pro vykreslení bitmap souběžně s referencí na bitmapu (viz datový typ bmp).

Rozdělení zadání bitmapy na dvě části (pevná část typu **bmp** v kódové paměti a proměnná část typu **bmpmode** v datové paměti) je velmi užitečné - jednu pevně definovanou bitmapu je pak totiž možné dynamicky zobrazovat různými způsoby, jen změnou položek proměnné **bmpmode.**

Použití :	**ve funkcích pracujících s bitmapami**
Definice :	type struct pxy org, ; x-y souřadnice umístění bitmapy (v pixelech) pxy sz, ; x-y vyhrazený rozměr pro bitmapu (v pixelech) byte pn, ; č.zobrazovaného obrázku (má-li jich bitmapa více) byte mode ; mód zobrazení (otočení, zrcadlení....) end **bmpmode**

org

Umístění bitmapy. Změnou tohoto parametru s ní lze pohybovat po displeji.

sz

Rozměr bitmapy. Bitmapa sice má svůj pevný rozměr zadefinovaný již v konstantě typu bmp, parametrem **sz** se ale dá nastavit i menší rozměr (pak bude zobrazen jen výřez bitmapy), nebo větší rozměr (bude buď zobrazena normálně, nebo v režimu "tile" s ní může být tato plocha vyplněna - viz dále).

pn

Číslo obrázku 0..N. Pokud má bitmapa více obrázků, tímto parametrem se vybere obrázek, který se bude zobrazovat.

mode

Různé volby zobrazení bitmapy. V bitovém vyjádření:

bit:	7	6	5	4	3	2	1	0
název:	- - -	- - -	- - -	TILE	MY	MX	R180	R90
význam:				tile	mirror		rotate	

Otočení bitmapy:

R90=0, R180=0	0°	výchozí stav, bez otočení
R90=1, R180=0	90°	doleva (proti směru hodinových ručiček)
R90=0, R180=1	180°	doleva (proti směru hodinových ručiček)
R90=1, R180=1	270°	doleva (proti směru hodinových ručiček)

Zrcadlení (převrácení) bitmapy:

MX=0, MY=0	výchozí stav, bez převracení
MX=1, MY=0	převrácení/zrcadlení dle svislé osy (co bylo vpravo, je vlevo a naopak)
MX=0, MY=1	převrácení/zrcadlení dle vodorovné osy (co bylo nahoře, je dole a naopak)
MX=1, MY=1	převrácení/zrcadlení v obou osách (kombinace obojího)

Vyplnění bitmapou ("vydláždění"):

TILE=0	výchozí stav, bitmapa se ve vyhrazeném prostoru zobrazí jen 1x
TILE=1	bitmapou se "vydláždí" vyhrazený prostor (bitmapa se vykreslí tolikrát, jak je třeba)

Pro zadání parametru mode lze použít i předdefinované konstanty:

Otočení 0°:	**_bmp_00**	(0x00)	Zrcadlení vlevo/vpravo:	**_bmp_mx**	(0x04)
Otočení 90°:	**_bmp_90**	(0x01)	Zrcadlení nahoře/dole:	**_bmp_my**	(0x08)
Otočení 180°:	**_bmp_90**	(0x02)			
Otočení 270°:	**_bmp_90**	(0x03)	Režim dláždění:	**_bmp_tile**	(0x10)

TIP:

Různorodé využití nabízí režim "tile", kdy se danou bitmapou "vydláždí" celý obdélník o rozměru **sz**. Takto se i s velmi malou bitmapou dá docílit efektních výsledků - např. při vyplňování ploch texturami, nebo např. pro kreslení různých liniových objektů (potrubí apod.) s různě vykresleným profilem. Pro libovolné potrubí stačí udělat jen několik čtvercových bitmap, typicky pro rovný úsek a zahnutý pro koleno. Jejich dlážděním vedle sebe a různým otáčením a zrcadlením kolen vznikne požadovaný tvar. A pokud mají bitmapy více obrázků, je možno udělat i snadnou animaci např. proudícího média v potrubí.

Příklad nastavení pro vyplnění obdélníku svisle zrcadlenou bitmapou:

```
;------------------------- definice proměnné typu bmpmode
var bmpmode SetBmp
;------------------------- nastavení parametrů zobrazení bitmapy
SetBmp.org.x = 50
SetBmp.org.y = 20
SetBmp.sz.x = 100
SetBmp.sz.y = 65
SetBmp.pn = 0
SetBmp.mode = _bmp_mx + _bmp_tile
;------------------------- vykreslení bitmapy BMP_voda dle parametrů
GdiBmp(BMP_voda, SetBmp)
```

font

Detailní znalost této struktury není při tvorbě programů nutná.

Strukturu generuje nástroj pro začlenění fontů do programu. Je využitelná pouze pro konstanty, nelze ji použít např. pro proměnné vytvořené uživatelem v datové paměti. Strukturovaná konstanta typu **font** je de-facto součástí vygenerovaného fontu a její název reprezentuje daný font v uživatelském programu (a zadává se coby reference na font do funkce GdiSetFont).

Konstrukce fontu viz popis funkce GdiSetFont.

Použití :	**pro zadefinovaní dalšího fontu do systému**
Definice :	type struct byte width, ; vyhrazená šířka znaku (v pixelech) byte height, ; vyhrazená výška znaku (v pixelech) string ptr ; odkaz na data fontu end **font**

linept

Slouží pro definici úseku buď při kreslení lomené čáry nebo kreslení nepravidelného N-úhelníku.

Obsahuje jednak položky x,y pro zadání relativní souřadnice (podobně jako pxy), jednak navíc položku **mode**, která říká zda jde o kreslený úsek s čárou, nebo prázdný úsek mezi dvěma čarami.

Pro zadávání hodnoty mode lze též využít předdefinované konstanty _moveto a _lineto.

_moveto (mode = 0) prázdný úsek, kreslicí pero se sem posune z předešlého bodu

_lineto (mode = 1) plný úsek, kreslicí pero sem nakreslí čáru z předešlého bodu

Použití :	**ve funkcích GdiBeginPoly a GdiPoly pro kreslení lomené čáry** **jako prvek pole struktury nangle pro funkce GdiNAngle / GdiFillNAngle**
Definice :	type struct int x, ; souřadnice x int y, ; souřadnice y byte mode ; typ spojnice: přesun/kreslení end **linept**

nangle

Slouží pro uložení celé sekvence úseků pro konstrukci N-úhelníku, nebo jiných objektů tvořených lomenou čarou. Je to de-facto pole úseků typu linept, s upřesněním počtu platných úseků. Má kapacitu na maximálně 256 zlomových bodů.

Používá se pro funkce GdiNAngle a GdiFillNAngle.

Použití :	**pro definici tvaru N-úhelníku pro funkce GdiNAngle / GdiFillNAngle**
Definice :	type struct byte len, ; počet platných bodů linept[256] pt ; pole úseků typu **linept** end **nangle**

arcmode

Popisná struktura pro definici tvaru oblouku nebo kruhové výseče. Hodnoty úhlů jsou chápány v desetinách úhlového stupně tj. kružnice odpovídá úhlu 3600 (a do tohoto rozsahu jsou všechny zadávané hodnoty úhlů interně převáděny). Pozice 0.0° je definována na kružnici vpravo (směr východ, resp. pozice 3:00 hod. na hodinovém ciferníku). Kladný úhel je chápán ve směru proti pohybu hodinových ručiček z pozice 0° a záporný naopak. Např. úhel 270.0° lze vyjádřit hodnotou 2700, nebo -900 (nebo také 6300, -4500, atd...).

Oblouk, resp. výseč, se kreslí vždy od počátečního bodu ve směru proti pohybu hodinových ručiček. Poloměr opisované kružnice je dán pouze parametrem **radius.x,** druhá složka souřadnice - radius.y musí mít nenulovou hodnotu!

Použití :	**definice tvaru kruhového oblouku pro funkci GdiArc** **definice tvaru kruhové výseče pro funkci GdiSlice** **definice tvaru plné kruhové výseče pro funkci GdiFillSlice**
Definice :	type struct int sangle, ; úhel přímky vedené počátečním bodem vzhledem k 0° int eangle, ; úhel přímky vedené koncovým bodem vzhledem k 0° pxy radius ; poloměr kružnice oblouku end **arcmode**

_uart

Datová struktura _uart se předává procedurám UartConfig, UartTxR a UartStopTxR pro ovládání komunikace na lince RS485 automatu.

Použití :	**ve funkcích UartConfig / UartTxR / UartStopTxR** **pro nastavování a vyhodnocování komunikace na lince RS485**
Definice :	type struct word cfg, ; volba linky RS485, komunikační parametry longword bdrate, ; komunikační rychlost byte d9b_mode, ; volba módu 9. bitu word tx_len, ; počet znaků k odvysílání word rx_len, ; počet očekávaných znaků pro příjem longword rx_timeout1, ; čas 1 vypršení transakce longword rx_timeout2, ; čas 2 vypršení transakce word rx_n, ; počet přijatých znaků v transakci byte rx_frmerr, ; indikace chyby parity / chyby 9. bitu v transakci byte rx_pause, ; nepoužívaná položka byte status, ; stav linky / ovladače _uart_buffer tx_buf, ; pole 512 bajtů pro vysílané znaky _uart_buffer rx_buf ; pole 512 bajtů pro přijímané znaky end **_uart**

Jednotlivé položky struktury jsou:

cfg

Nastavuje se v programu před předáním proměnné proceduře UartConfig, slouží tedy k předání parametrů komunikace. Význam jednotlivých bitů hodnoty je následující:

bit 0 ... počet stop-bitů - hodnota {0;1} znamená volbu {1;2} stopbity

bit 1 ... 9-bitový mód - hodnota {0;1} znamená volbu {8;9} datových bitů ve znaku (při komunikaci s paritou se také paritní bit počítá mezi datové)

bit 2 ... komunikace s paritou - hodnota {0;1} znamená volbu {bez parity;s paritou}

bit 3 ... volba parity - hodnota {0;1} znamená volbu {sudá;lichá} parita

bity 6 až 4 ... číslo linky RS485 - pro automat MPC400 lze zadat pouze jednu z hodnot {0;1}, ty odpovídají volbě linky {L1;L2}, resp. volbě požadovaného ovladače {USART0;USART1}

Pozn.: V případě použití nastavení 8 datových bitů a povolení použití parity bude na linku odesíláno pouze spodních 7 bitů každého datového bajtu. Přijaté datové bajty pak budou mít vždy 8. bit v hodnotě 0.

bdrate

Nastavuje se v programu před předáním proměnné proceduře UartConfig, slouží tedy k předání parametrů komunikace. Má význam rychlosti komunikace na lince v jednotkách Bd. Povolený rozsah hodnot je 1000 až 230400.

d9b_mode

Nastavuje se v programu před zahájením transakce, ale lze měnit i v průběhu transakce. Slouží k nastavení módu 9. bitu při komunikaci s 9 datovými bity bez parity, v ostatních konfiguracích se neuplatní. Nastavení hodnoty {0;1} udává přímo hodnotu 9. bitu {0;1}. Hodota různá od {0;1} značí, že 9. bit prvního znaku má být 1, ve zbylých znacích má být 0. V průběhu vysílání ovladač nastavuje 9. bit znaků podle zvoleného módu. Při příjmu znaků ovladač kontroluje shodu 9. bitu s očekávanou hodnotu, v případě neshody nastaví příznak *.rx_frmerr*.

tx_len

Nastavuje se v programu před zahájením transakce, udává počet znaků z *.tx_buf* k odvysílání. Lze zadat číslo maximálně do délky *.tx_buf* (tj. 512 znaků). Při zadané hodnotě 0 bude vysílání vynecháno.

rx_len

Nastavuje se v programu před zahájením transakce, udává očekávaný počet přijímaných znaků. Lze zadat libovolné číslo, i větší než je délka *.rx_buf* (tj. 512 znaků). Při zadané hodnotě 0 bude

příjem vynechán. Po příjmu zadaného počtu znaků bude transakce automaticky ukončena.
V případě, že je zadána hodnota větší než 512 a dojde k příjmu 512 znaků, bude transakce ukončena s výsledkem _ust_rxfull. Když u transakce není předem známo, kolik znaků bude příchozí datový rámec obsahovat, zadáme raději časové kritérium ukončení transakce, pomocí *.rx_timeout2*. Pak je vhodné nastavit *.rx_len* na libovolnou hodnotu větší, než je největší možná délka příchozího rámce.

rx_timeout1

Nastavuje se v programu před zahájením transakce, ale lze měnit i v průběhu transakce. Udává dobu v jednotkách 0,1ms počítanou od zahájení příjmu, během níž musí dojít k příjetí prvního znaku. Po zadané době jinak transakce skončí stavem _ust_timeout1. Při zdané hodnotě 0 nebude timeout vyhodnocován. Pro správné fungování komunikace je třeba nastavit timeout minimálně na dobu trvání jednoho znaku při nastavené komunikační rychlosti a počtu bitů na znak, a s rezervou 0,1ms. Dále je třeba počítat s časem na zpracování a zahájení vysílání na straně zařízení na lince. Tento čas se nicméně většinou nastavuje v řádu doby trvání několika jednotek nebo desítek znaků.

Dobu trvání jednoho znaku v milisekundách lze spočíst vztahem *1000*N/R*, kde *N* je počet bitů na znak a *R* je komunikační rychlost v baudech (Bd). Hodnota *N* bude podle nastavených komunikačních parametrů 10, 11, nebo 12 (1 startbit, 8-9 datových bitů, 1-2 stopbity).

rx_timeout2

Nastavuje se v programu před zahájením transakce, ale lze měnit i v průběhu transakce. Udává dobu v jednotkách 0,1ms počítanou od času naposledy přijatého znaku (počínaje příjmem prvního znaku), během níž musí dojít k příjmu dalšího znaku. Po zadané době jinak transakce končí stavem _ust_timeout2. Při zdané hodnotě 0 nebude timeout vyhodnocován. Pro správné fungování komunikace je třeba nastavit timeout minimálně na dobu trvání jednoho znaku při zadané komunikační rychlosti a počtu bitů na znak, a s rezervou 0,1ms. Např. u protokol Modbus RTU je tento čas stanoven na dobu trvání 3/2 znaku.

rx_n

Hodnota je nastavována ovladačem v průběhu transakce. Udává počet doposud přijatých znaků, resp. celkový počet přijatých znaků po ukončení transakce.

rx_frmerr

Hodnota je ovladačem nastavena na 0 na začátku transakce, v průběhu transakce ji může ovladač změnit na 1. Hodnota 1 indikuje detekci chyby paritního bitu u některého ze znaků v případě konfigurace s paritou. V případě konfigurace 9 datových bitů bez parity hodnota 1 indikuje detekci chyby v 9. bitu u některého ze znaků (9. bit neodpovídá hodnotě podle *.d9b_mode*).

status

Hodnota je nastavována ovladačem v průběhu transakce, případně v průběhu procedur UartConfig, UartTxR a UartStopTxR. Udává stav naposledy zadané transakce, indikuje chybu parametrů transakce, případně indikuje chybějící ovladač USART. Pro většinu platných hodnot jsou předdefinovány symbolické konstanty.

0 (_ust_none) …	parametry předávané proměnnou struktury _uart jsou neplatné
1 (_ust_running) ...	transakce probíhá
2 (_ust_done) ...	poslední transakce ukončena příjmem zadaného počtu znaků
3 (_ust_rxfull) ...	poslední transakce ukončena příjmem maximálního možného počtu znaků při nastavení na příjem většího než maximálního možného počtu znaků
4 (_ust_userstop) ...	zadána nová konfigurace / poslední transakce zastavena z programu
5 (_ust_timeout1) ...	poslední transakce ukončena timeoutem 1
6 (_ust_timeout2) ...	poslední transakce ukončena timeoutem 2
8 ...	v automatu není spuštěn požadovaný ovladač USART

tx_buf

Pole pro zápis znaků k odvysílání délky 512, příslušný počet znaků je třeba zapsat v programu před zahájením transakce. Vysílání proběhne postupně od první položky pole.

rx_buf

Pole s přijatými znaky délky 512. Přijímané znaky jsou ovladačem zapisovány postupně od začátku pole tak, jak v průběhu transakce přicházejí.

_uart_crc

Slouží k předání parametrů výpočtu funkcím CRC16_RS a CRC16_LS.

Použití :	**definice parametrů pro výpočet CRC-16 funkcemi CRC16_RS / CRC16_LS**
Definice :	type struct word poly, ; dělící polynom výpočtu CRC-16 word crc ; inicializační hodnota výpočtu CRC-16 end **_uart_crc**

dataptr

Slouží k předání parametrů systémovým funkcím pro obecné zpracování dat. Jednotlivé položky struktury nejsou programově přístupné. Inicializace datového typu dataptr je možná pouze pomocí

zavedeného operátoru reference @. Použití datového typu dataptr v jazyce Simple má omezení, která jsou vynucena požadavkem na bezpečnost použití ukazatele.

Použití :	**datový typ představující bezpečný typ ukazatele**
Definice :	type struct longword addr, ; adresa na předávaná data longword size ; velikost datového prostoru pro data v bytech end **dataptr**

S datovým typem dataptr jsou povoleny operace:

null …. datový ukazatel, který neukazuje na žádnou proměnnou nebo konstantu (předdefinovaný symbol)

deklarace … deklarace může být v datové i kódové paměti

inicializace … inicializaci proměnné typu dataptr provádíme pomocí operátoru reference @

Příklad inicializace datového typu dataptr a použití hodnoty null

```
code longword k_lw = 4256
code byte k_byte = 42
;----------------------------------------
code dataptr reference_k = @k_lw
table dataptr[3] = (@k_lw,@k_byte,null)
;----------------------------------------
var dataptr prvni
var dataptr druhy
var dataptr treti
var byte val_byte
var longword val_lw
;----------------------------------------
prvni = @val_byte
druhy = @val_lw
treti = null
```

přiřazení a předávání parametrů…. datový ukazatel je možné přiřazovat a předávat jako parametr

porovnání … proměnnou typu dataptr je možné porovnávat na rovnost nebo nerovnost s jinou proměnou typu dataptr nebo hodnotou null. Ostatní typy porovnání nejsou podporovány. V operacích porovnání je porovnávána pouze položka **addr** tj. adresa.

Příklad použití datového typu dataptr

```
code string name = "jmeno"
code dataptr nameptr = @name
var dstring text
;----------------------------------------
var dataptr muj
```

```
var dataptr tvuj
var dataptr[30] pole
var byte idx
;----------------------------------------
pole[idx] = @idx
pole[5] = @idx
muj = @idx
tvuj = muj
if muj = tvuj then muj = tvuj ; porovnává se pouze adresa
if muj = null then muj = tvuj
if muj = @idx then muj = tvuj
if @idx <> muj then muj = tvuj
;----------------------------------------
Copy(@text,@jmeno)      ; string -> dstring pomocí operátoru @
Copy(@text,nameptr)     ; string -> dstring přes dataptr
```

_circular_buffer

Slouží k uchování dat při kontinuálním zpracování datových přenosů. Systémové funkce s touto strukturou pracují automaticky, takže není nutné ji v aplikaci nějak zpracovávat.

Použití :	**kontinuální zpracování dat při datových přenosech**
Definice :	type struct word wi, ; zápisový index word ri, ; čtecí index byte[512] buf ; pole bajtů pro uchování dat end **_circular_buffer**

_scan_value

Slouží k uložení návratové hodnoty při iterativním zpracování textu při datových přenosech.

Použití :	**Iterativní zpracování textů při datových přenosech**
Definice :	type struct longword lw, ; číslo ve tvaru neznaménkové hodnoty 32bitů longint li, ; číslo ve tvaru znaménkové hodnoty 32bitů float f, ; číslo ve tvaru float dstring s, ; hodnota v textovém tvaru byte valid ; označuje, které položky obsahují platnou hodnotu end **_scan_value**

valid

Hodnota určuje, které položky struktury obsahují platná data.

bit 0 ... položka lw obsahuje platnou hodnotu
bit 1 ... položka li obsahuje platnou hodnotu
bit 2 ... položka f obsahuje platnou hodnotu
bit 3 ... položka s obsahuje symbol nebo text
bit 4 ... položka lw obsahuje kód řídicího znaku např. separátor, operátor atp.
bity 7 až 5 ... jsou nevyužity

_scan_symbol

Umožňuje uživatelsky nastavit způsob iterativního zpracování textů např. řízení funkce pro převod na textu na velká písmena, zadání oddělovačů symbolů nebo označení pro zpracování nevýznamných znaků. V naprosté většině případů není použití specifického nastavení pro zpracování textů nutné, protože nejčastěji používaná pravidla jsou přímo součástí systému a volí se pouze parametrem zpracování. Pole je při zpracování používáno způsobem index→hodnota, kdy index představuje kód právě zpracovávaného znaku a hodnota je následně použita pro další vyhodnocení a zpracování znaku.

Použití :	**řízení Iterativního zpracování textů při datových přenosech**
Definice :	type byte[256] **_scan_symbol** ; řízení zpracování textů

Každá položka pole obsahuje jednu z následujících hodnot:

0 **... tr_white_marker** - označuje pro zpracování nevýznamný znak např. mezera má kód 32 a je nevýznamným znakem, proto bude mít položka pole _scan_symbol[32] hodnotu 0 (tr_white_marker)

1 ... **tr_sign_char -** označuje, že znak je pro zpracování textu významným znakem označující separátor, operátor, který je důležitý pro zpracování textu

32 - 256 **... tr_char_marker** hodnoty označují znak, který může být součástí textového symbolu. Hodnota je současně použita při záměně malého písmena za velké v případě, že zpracování textu není citlivé na velká a malá písmena. Například položka pole s indexem 61 odpovídá malému písmenu „a“ bude mít hodnotu 41. Hodnota 41 odpovídá velkému písmenu „A“. Proto bude v případě převodu malých písmen na písmena velká nahrazen kód 61 kódem 41 tj. malé „a“ bude převedeno na velké „A“. Naopak číslice 0 má index 30 a neexistuje malá a velká nula, a tak bude mít položka pole s indexem 30 hodnotu 30.

_wr_buffer

Řídicí automat spravuje provoz na sběrnici EXBUS a řeší přenos dat přes všechny typy IO uzlů. Uzel typu WR_BUFFER slouží k přenosu dat s potvrzením o jejich doručení ve směru od řídicího automatu do rozšiřující jednotky. Rychlost přenosu dat je v daném případě přizpůsobena rychlost zpracování na straně periferie.

Použití :	**IO uzel s potvrzením přenosu dat, směr řídicí automat rozšiřující jednotka**
Definice :	type struct byte[32] data, ; pole bajtů pro obsah zprávy (data) byte idx, ; značka obsahu dat bit stbsy, ; řídicí bit START/BUSY byte len, ; počet platných bajtů v poli data byte time, ; max. čas pro přenos dat v desítkách ms byte[4] nu ; nepoužito - rezerva end **_io_buf**

Z hlediska uživatelského programu je ovládání uzlu jednoduché. Nejprve otestujeme zda je uzel připraven přijmout nová data. To je tehdy, pokud je řídicí bit STBSY nastaven na hodnotu 0. V případě, že je bit nastaven na hodnotu 1, probíhá právě předchozí přenos a hodnoty v datových položkách uzlu nelze měnit.

Je-li uzel zel připraven přijmout nová data (STBSY = 0), pak nejprve zapíšeme hodnoty do položek data, idx, len. V případě, že jsme data zapsali, zahájíme přenos tak, že nastavíme bit STBSY na hodnotu 1 tj. na START a spustíme tak žádost o vyslání dat. Od této chvíle je zmíněný bit STBSY ve funkci BUSY a ukazuje nám, zda je uzel zaneprázdněn přenosem dat, či zda přenos skončil. V případě, že přenos probíhá je bit STBSY nastaven na 1. Po ukončení přenosu se bit nastaví zpět na 0 a uzel je připraven přijmout další data ke zpracování.

Na řízení přenosu má vliv též položka time. Tato položka může obsahovat hodnotu v rozsahu 0-127, přičemž tato hodnota značí čas, který je vyhrazen pro realizaci přenosu. Pokud v tomto čase není přenos realizován, dojde k chybě a systém nastaví bit STBSY na hodnotu 0 a položku len též na hodnotu 0. Tím se dává najevo, že přenos nebyl uskutečněn a že nebyl přenesen žádný byte. Hodnota položky time se zadává v desítkách milisekund. Pokud je tedy hodnota nastavena na 100, je vyhrazen pro přenos čas 1s.

Důvodem pro zavedení položky time, je fakt, že na systémové úrovni běží po lince EXBUS dotazy na zpracování přenesených dat. Pokud tedy periferie přenesená data nepřevezme, není možné začít přenášet data nová. Tím pádem by mohla nastat situace, že by bit STBSY byl neustále nastaven, program by čekal na dokončení přenosu, k čemuž by vlastně, díky nepřevzetí předešlých dat periferií, nikdy nedošlo. Proto se tento problém řeší položkou time.

_rd_buffer

Uzel typu RD_BUFFER je určen pro přenos dat s potvrzením ve směru od periferie po řídicí automat. Princip komunikace je obdobný jako v případě uzlu WR_BUFFER. Uživatelský program v řídicím automatu sleduje bit STBSY.

Pokud detekuje nastavení tohoto bitu na 1, znamená to, že byl zahájen přenos dat a že data v položkách data, idx a len jsou platná. V závislosti na komunikačním protokolu a významu přijatých dat, uživatelský program data zpracuje a nastaví bit STBSY do 0. Tím ukončí stav čekání (BUSY)

na převzetí dat a uvolní tak uzel k dalšímu přenosu. Tomto případě není použita položka time, protože na straně přijímací nemá význam. O tom kdy data převezme rozhoduje program aplikace. Periferie data pouze ke zpracování pouze dodává.

Použití :	**IO uzel s potvrzením přenosu dat, směr řídicí automat rozšiřující jednotka**
Definice :	type struct byte[32] data,; pole bajtů pro obsah zprávy (data) byte idx, ; značka obsahu dat bit stbsy, ; řídicí bit START/BUSY byte len, ; počet platných bajtů v poli data byte[5] nu ; nepoužito - rezerva end **_io_buf**

Uzel typu RD_BUFFER je určen pro přenos dat s potvrzením ve směru od periferie po řídicí automat. Princip komunikace je obdobný jako v případě uzlu WR_BUFFER. Uživatelský program v řídicím automatu sleduje bit STBSY. Pokud detekuje nastavení tohoto bitu na 1, znamená to, že byl zahájen přenos dat a že data v položkách data, idx a len jsou platná. V závislosti na komunikačním protokolu a významu přijatých dat, uživatelský program data zpracuje a nastaví bit STBSY do 0. Tím ukončí stav čekání (BUSY) na převzetí dat a uvolní tak uzel k dalšímu přenosu. Tomto případě není použita položka time, protože na straně přijímací nemá význam. O tom kdy data převezme rozhoduje program aplikace. Periferie data pouze ke zpracování pouze dodává.

PNET_STATUS, SPNET_STATUS

Použití :	**informace o aktivních zařízeních na lince PESNET a SIMPLENET**
Definice :	longword - chápán jako pole bitů, kdy každý bit reprezentuje jednu aktivní stanici na lince PESNET a SIMPLENET, každý bit proměnné tak odpovídá jedné adrese na lince, pokud se jedná o bit s vlastní adresou zařízení, je tento bit nevyužit a jeho hodnota je trvale 0

Řada 400 – programovací příručka - technický list 5.2017
verze 2.0
platné pro verze hardware od r.v.2015

http:// www.micropel.cz info@micropel.cz