# KIT-BASIC

# Definice jazyka

***SofCon®*** s.r.o
Střešovická 49
162 00 Praha 6
tel/fax: (02) 20 180 454

# OBSAH

# 1. KIT-BASIC - Definice jazyka

## 1.1 Definice kostry programu KIT-BASICu:

```
<program>              ::=  {<declaration_part>}
<declaration_part>     ::=  <options_declarations>                  definice options překladače
                            <define_declaration>              definice změny lexikálního elementu
                            <constant_declarations> |            definice identifikátorů konstant
                            <symbol_declarations> |           definice symbolických názvů registrů
                            <config_declarations> |               definice konfigurací LP-objektů
                            <terminal_declaration> |      definice vstupů a výstupů konc. zařízení
                            <procedure declaration>                          definice procedury
```

*Jednotlivé výše uvedené deklarace se mohou vyskytovat v libovolném pořadí, libovolněkrát za sebou. Musí být dodržena zásada, že všechny prvky jazyka, dané svými identifikátory, musí být nejprve definovány a teprve potom mohou být použity. Program musí povinně absahovat alespoň proceduru s povinným názvem MAIN (základní algoritmus).*

Některé další identifikátory procedur mají speciální význam. Procedura INIT se provádí po inicializaci HW, procedura FAST se volá automaticky každých 10, resp. 20, 50 ms. Procedura WAIT je nadefinována systémem a smí být pouze volána.

Na libovolném místě programu může být uveden komentář v komentářových závorkách { } resp. (* *).

Rozložení programu do jednotlivých řádek je libovolné (volný formát zápisu).

Malá a velká písmena se od sebe při psaní programu nerozlišují (vyjma stringu)

**Příklad:**

```
{ Program PRVNI }
{ ---------------- Definice konstant ---------------------------------------------------- }
constant
  PRVNISMER=1;                                                    { symbolicky nadefinovaná konstanta }
{ ---------------- Definice symbolických názvů registru ------------------------------- }
symbol
  CITAC=I200;                                                  { symbolické označení pro registr I200 }
  SMER=I202;                                                   { symbolické označení pro registr I202 }
{ ---------------- Definice objektu ----------------------------------------------------- }
configuration
  SWOBJ=TERM10, ADR=$2300, VAR=B100:5;
{ ---------------- Definice obrazovky č.0 na terminálu TERM10 ------------------------- }
terminal TERM10:0;
  begin
  font 1; position 20,20; print "CITAC=",CITAC:5;              { výpis registru CITAC na obrazovku }
          position 20,30; print "SMER= ",SMER:5;               { výpis registru SMER na obrazovku }
  onkey
    '+':SMER:=1;                        { po stisku klávesy "+" resp. "-" změna hodnoty registru SMER }
    '-':SMER:=-1;
    end;
  end;
{ ---------------- Definice procedury, která se provede 1x po RESETu ----------------- }
procedure INIT;
  begin SMER:=PRVNISMER; end;          { po RESETu má SMER hodnotu +1 (ostatní registry vynulovány) }
{ ---------------- Definice procedury, která se provádí v intervalu 10ms ------------- }
procedure FAST;
  begin CITAC:=CITAC+SMER; end;  { vždy 1x za 10ms se registr CITAC změní o hodnotu v registru SMER }
{ ---------------- Definice hlavního algoritmu, prováděného cyklicky ------------------}
procedure MAIN;
  begin
  TERM10_Led:=CITAC/100;                    { změna portu LED na TERM10 cyklicky dle registru CITAC }
  end;
```

## 1.2 Definice základních lexikálních elementů:

*Uživatelské datové struktury:*
1) Společná datová struktura obsahuje registry RegV (1 bit), RegB (1 byte), RegI, RegW (2 byty), RegL (4 byty), RegS (standardně 20 byte, 1.byte=délka, max.19 znaků, velikost možno změnit při symbolické definici RegS v sekci **symbol**). U ostatních registrů je možno stejným způsobem definovat blok registrů daného typu, který se potom může používat a indexovat jako pole v jazyku pascal, jeho počátek je v registru, kterým se blok definuje.
*2) Datová struktura reálných čísel RegR*
*Záleží plně na uživateli, zda se jednotlivé používané registry v paměťovém prostoru překrývají nebo ne.*
*3) Systémové registry*
*Speciální registry, které obsluhuje systém, zapisuje do nich datum, čas, verzi překladače, interpretu a programu, nastavení rychlosti procedury sysfast, současnou délku a maximální délku vykonání procedury main, maximální délku vykonání procedury fast(10) a její přetečení, stav systémové komunikace - její běh, zinicializování a flag jestli je master, uživatel na ně může přistupovat přes již nadefinované symbolické názvy SYSYEAR, SYSMONTH, SYSDAY, SYSDOW, SYSHOUR, SYSMIN, SYSSEC, SYSPACKTIME, SYSPROCVER, SYSCOMPVER, SYSPROGVER, SYSFASTFREQ, SYSMAINTIME, SYSMAINMAXTIME, SYSFAST10MAXTIME, SYSFAST10OVERFLOW, SYSCOM_STATE, SYSCOM_INI, SYSCOM_MASTER*

**<sysregisters>** ::= **SYSYEAR** | **SYSMONTH** | **SYSDAY** | **SYSDOW** | **SYSHOUR** | **SYSMIN** | **SYSSEC** | **SYSPACKTIME** | **SYSPROCVER** | **SYSCOMPVER** | **SYSPROGVER** | **SYSFASTFREQ** | **SYSMAINTIME** | **SYSMAINMAXTIME** | **SYSFAST10MAXTIME** | **SYSFAST10OVERFLOW** | **SYSCOM_STATE** | **SYSCOM_INI** | **SYSCOM_MASTER**

**<any_character>** ::= libovolný znak standardu ASCII
**<letter>** ::= **'A'..'Z'** | **'a'..'z'** | **'_'** |**'\'**]
**<digit>** ::= **'0'..'9'**
**<value>** ::= <digit> {<digit>}
**<decvalue>** ::= <value>
**<bitNo>** ::= 0..15 (decimálně zadané číslo)
**<hexdigit>** ::= **'0'..'9'** | **'A'..'F'** | **'a'..'f'**
**<hexvalue>** ::= <hexdigit> {<hexdigit>}
**<ASCvalue>** ::= **'<**any_character**>'** libovolný znak v jednoduchých uvozovkách
**<spckeyvalue>** ::=
**zCr**|**zBs**|**zSBs**|**zEsc**|**zSpace**|**zDel**|**zUp**|**zDn**|**zLe**|**zRi**|**zF1**|**zF2**|**zF3**|**zF4**|**zF5**|**zF6**|**zF7**|**zF8**|**zF9**|**zF10**|**zHome**|**zPgUp**|**zEnd**|**zPgDn**|**zIns**|**zTab**

**<RegType>** ::= **BYTE** | **WORD** | **INTEGER** | **LONGINT** | **DATETIME** | **REAL** | **STRING**
**<AutoRegister>** ::= <RegType>
**<CastType>** ::= <RegType>

**<keyvalue>** ::= <decvalue> | <hexvalue> | <ASCvalue> | <spckeyvalue>
**<string_const>** ::= <string_element> { + <string_element> } *stringové elementy spojené "+"*
**<string_element>** ::= " <plain_text> " | **#<**number**>** | <string_element><string_element>
*posloupnost znaků ve dvojitých uvozovkách, nebo tzv. křížkové konvence, které na sebe mohou (bez mezery !) navazovat*
**<identifier>** ::= <letter> { <letter | <digit> } místo, kde je identifikátor použit
**<identifier_def>** ::= <identifier> *místo, kde je identifikátor definován*

*Identifikátor může mít lib. počet znaků, rozlišuje se však pouze podle prvních 16 znaků.*
*Žádný uživatelem nadefinovaný identifikátor nesmí být shodný s implicitně nadefinovanými identifikátory:*
*- absolutní jména všech registrů (R11, B10, B2.2 atd.)*
*- klíčová slova, identifikátory standardních procedur a funkcí (viz kap. 0)*
*Všechny uživatelské identifikátory jsou nadefinovány globálně v celém programu.*

**<float_number>** ::= <decvalue>.<decvalue> | <const_name> *reálné číslo bez znaménka*
**<decnumber>** ::= <decvalue> *celé číslo dekadicky bez znaménka*
**<hexnumber>** ::= **$** <hexvalue> *celé číslo hexadecimálně bez znaménka*
**<number>** ::= <decnumber> | <hexnumber> | <const_name> *celé číslo bez znaménka*
**<sign_number>** ::= <signum> <number> *celé číslo*
**<length>** ::= <const_expression> *délka stringového registru, nebo délka bloku*

**<register>** ::= <Reg_ALL> | <sysregisters>
*libovolný typ registru vyjádřený absolutně nebo symbolicky*

**<symbolregister>** ::= <symbolregister_casted> | <symbolregister_auto>
*libovolný typ registru vyjádřený absolutně*

**<symbolregister_simple>** ::= <register> | *libovolný typ registru vyjádřený absolutně*
<Reg_ALL><signum><const_expression> | a *indexovaně*
<Reg_ALL>**[**<const_expression>**]**

**<symbolregister_casted>** ::= <symbolregister_simple> | <CastType>(<symbolregister_casted>)
*nadefinování registru s přetypováním - je povoleno pouze přetypování na menší typ*

**<symbolregister_auto>** ::= <AutoRegister> | *automatické vyhledání prvního volného registru*
<AutoRegister> **:** <const_expression>
*automatické vyhledání s alokováním zadaného počtu registrů za vyhledaným registrem ( vhodné pro práci s registry jako s jednorozměrným polem )*

**<IndexRegister>** ::= <register> | <register>**[**<expression>**]**
*libovolný typ registru vyjádřený pomocí indexu - výrazem*

**<IndexRegister2>** ::= <register> | <register>**[**<register>**]** | <register>**[**const_expression>**]**
*libovolný typ registru vyjádřený pomocí indexu - registr nebo konstanta*

**<regIndexNum>** ::= <RegNum>**[** RegNum **]**
číselný *registr vyjádřený pomocí indexu - z registru*
**<regIndexNumC>** ::= <RegNum>**[** RegNumC **]**
číselný *registr vyjádřený pomocí indexu - registru nebo číselné konstanty*

**<regNum>** ::= <regVBWILD> | <regR> *registr typu číselného*
**<regCmn>** ::= <regVBWILD> | <regS> *registr ze společného adres. prostoru*
**<regNumC>** ::= <regNum> | <const_expression> *registr číselného typu nebo celočíselná konstanta, může být daná i konstantím výrazem, v rozsahu -4095..4095*
**<regNumC_Index>** ::= <regNum> | <regIndexNumC> | <RegNumC>
*registr číselného typu nebo celočíselná konstanta v rozsahu -4095..4095*

**<regV>** ::= <RegB>**.**<BitNo> | <regW>**.**<BitNo> | <regV>
*registr typu bit, číslo 0-511, bit 0-7*
**<regB>** ::= **B**<value> | <Bsymbol_name> *registr typu byte, číslo 0-1999*
**<regW>** ::= **W**<value> | <Wsymbol_name> *registr typu word, číslo 0-1998*
**<regI>** ::= **I**<value> | <Isymbol_name> *registr typu integer, číslo 0-1998*
**<regL>** ::= **L**<value> | <Lsymbol_name> *registr typu longint, číslo 0-1996*
**<regD>** ::= **D**<value> | <Dsymbol_name> *registr typu datetime, číslo 0-1996*
**<regS>** ::= **S**<value> | <RegS>:<length>
*registr typu string, číslo 0-1999, číslo registru + délka stringu <length> nesmí překročit 2000*
**<regR>** ::= **R**<value> | <Rsymbol_name> *registr typu real, číslo 0-255*
**<Xsymbol_name>** ::= <symbol_name> *identifikátor,tj.symbolické jméno registru daného typu* registr *daného typu*

**<regBW>** ::= <regB> | <regW> *registr typu celočíselného nezáporného*
**<regBWIL>** ::= <regB> | <regW> | <regI> | <regL>
*registr typu celočíselného vyjma typu BIT*
**<regVBWIL>** ::= <regV> | <regB> | <regW> | <regI> | <regL> *registr typu celočíselného*
**<regVBWILD>** ::= <regV> | <regB> | <regW> | <regI> | <regL> | <regD> | <sysregisters>
*registr typu celočíselného s typem datetime*
**<Reg_ALL>** ::= <regV> | <regB> | <regW> | <regI> | <regL> | <regD> | <regR> | <RegS>

**<signum>** ::= **+** | **-** *znaménko plus nebo minus*
**<relational operators>** ::= **<=** | **<** | **<>** | **=** | **>=** | **>**
**<add_math_operators>** ::= **+** | **-**
**<add_log_operators>** ::= **xor** | **or**
**<add_operators>** ::= <add_math_operators> | <add_log_opeerators>
**<mult_math_operators>** ::= ***** | **/**
**<mult_log_operators>** ::= **and**
**<mult_shift_operators>** ::= **shl** | **shr**
**<mult_div_operators>** ::= **mod**
**<mult_operators>** ::= <mult_math_operators> | <mult_log_operators> |
<mult_shift_operators> | <mult_div_operators>
**<dummy>** ::= *nic*

**Příklady:**
**identifier:** Petr, \k, Dlouhy_rozdeleny_identifikator *Možné identifikátory*
**number:** 123, $123, *Číselné konstanty*

## 1.3 Deklarace Options:

**<option_declarations>** ::= **options** <option_definitions>
**<option_definitions>** ::= <option_definition> { ; <option_definition>}

**<option_definition>** ::= **FastFreq =** 10 | 20 | 50
*Frekvence proscesoru - konstanty 10, 20 nebo 50 (default 10)*
**ProgVer =** <number> *Verze programu - číslo 0-255; (default 255)*
**Gonio = Dec** | **Rad**
*Způsob vyhodnocování goniometcikých funkcí (sin, cos) - ve stupních nebo radianech.*
**OptCase =** <number>
*Zapnutí = 1 nebo vypnutí =0 optimalizace příkazu case tabulkou. Default hodnota je 1*
**Refresh =** <value>
Perioda prekreslovani displeje v ms , může nabývat hodnot 200 až 2500.
**UseSysCom =** 10 | 20
*Požadavek na uvolnění systemové seriové linky. Pokud do určité doby (specifikována v manuálu k Kit-Builderu) po resetu nedojde k navázání spojení s nadřízeným systémem.*
**DefStrLen** = <number>
*Standardní délka stringové proměnné (default=20); není dosud implementováno*

**<on_off>** ::= **on** | **off**

**Příklady:**
```
options
  FastFreq = 50 ;
  ProgVer = 20;
  Gonio = Dec;
option
  ProgVer = $1A;
```

## 1.4 Popis #DEFINE:

**<define_declaration>** ::= **#define** <element_name_def> <element_body>
**<element_name_def>** ::= <any_lexikal_element> *místo, kde je element definován*
**<element_body>** ::= <any_lexikal_element>
*Každý* **<element_name_def>** *lze nadefinovát jen jedenkrát.*

**<element_name>** ::= <any_lexikal_element> *místo, kde je nadefinovaný element použit*
*V další části programu se (vyjma komentářů) nahrazují nalezené* <element_name> *příslušným elementem* <element_body> *z* **<define_declaration>** *ještě před prováděním lexikální analýzy, tj. pomocí* **<define_declaration>** *můžeme předefinovat prakticky jakýkoliv lexikální element.*

**Příklady:**
```
#define proc procedure
#define \k onkey
```

## 1.5 Popis #INCLUDE:

**<include_declaration>** ::= **#include** <inc_program_name>
**<inc_program_name>** ::= <any_lexikal_element> *místo, kde je element definován*

**<include_declaration>** *místo, kde je otevřen soubor* **<inc_program_name>** *s příponou .pri a přidán ke kompilaci k hlavnímu zdrojovému souboru.*

**Příklad:**

```
#include objekty
```

# 1.6 Deklarace konstant:

*Sekce **constant** slouží k symbolickému nadefinování absolutní konstanty.*

```
<constant_declaration>     ::=  constant <constant_definitions>

<constant_definitions>     ::=  <const_definition> { ; <const_definition>}

<constant_definition>      ::=  <const_name_def> = <const_expression>
                                                    definice symbolické absolutní konstanty
<const_name_def>           ::=  <identifier>              místo definice absolutní konstanty

<const_name>               ::=  <identifier>               místo použití absolutní konstanty

<const_std_function>       ::=  sin (<const_expression>) |                                  sinus
                                cos (<const_expression>) |                                cosinus
                                ln (<const_expression>) |                  přirozený logaritmus
                                log (<const_expression>) |                 dekadický logaritmus
                                exp (<const_expression>) |                           exponent
                                sqrt (<const_expression>) |                         odmocnina
                                sqr (<const_expression>) |                    druhá mocnina
                                sgn (<const_expression>) |                           signum
                                abs (<const_expression>) |                 absolutní hodnota
                                min (<const_expression> , <const_expression>) |          minimum
                                max (<const_expression> , <const_expression>)            maximum
                                sizeof (<register>) | sizeof (<AutoRegister>)
                                                          velikost registru daného typu
```

Pozn.: Standardní konstantní funkce nejsou zatim implementovány.

```
<const_expression>         ::=  <const_simple> { <relational_operators> <const_simple> }
<const_simple>             ::=  <const_term> { <adding_operators> <constant_term> }
<const_term>               ::=  <const_faktor> { <multiplying_operators> <const_faktor> }
<const_faktor>             ::=  <decnumber> | <hexnumber> |                  celočíselná konstanta
                                <ASCvalue> |          celočíselná konstanta 0..255 vyjádřená znakově
                                <float_number> |                              reálná konstanta
                                ( <const expression> ) |                 libovolný konstantní výraz
                                <const_name> |                           symbolické jméno konstanty
                                <const_std_function> | volání vybrané standardní fce s konst. parametry
                                <signum> <const_faktor>                  unární operace "+" resp. "-"
```

*<const_expression> se vyhodnocuje při překladu. Pokud se ve výrazu vyskytuje Real, vyhodnocuje se nad typem Real, pokud ne, výraz se vyhodnocuje nad typem longint. Výsledný typ každé symbolické konstanty se určuje podle těchto pravidel = **TAB1:***

| způsob zadání <const_expression> resp. <expression> | rozsah 1.operandu | rozsah 2.operandu | typ výsledku |
|---|---|---|---|
| <dec_number>, <hex_number>, <ASC_value> | 0..255 | | byte |
| <dec_number>, <hex_number> | 256..65535 | | word |
| <dec_number>, <hex_number> | $2^{16}..2^{31}-1$ | | longint |
| <signum><dec_number> | $0..2^{15}-1$ | | integer |
| <signum><dec_number> | $2^{15}..2^{31}-1$ | | longint |
| <float_number> | real | | real |
| <signum><float_number> | real | | real |
| <string_const> | string | | string 1) |

*Pozn 1) Pouze pro <expression>, nepřípustné pro <const_expression> .*

**Příklad:**

```
constant

  PI=3.141592;
  PI2=2*PI;
  NESMYSL=(4+5*6)/PI;
  CTVRTY_BIT=1 shl 4;
  MASKA=$FE;
  DELKA=sizeof(W0);
```

## 1.7 Deklarace symbolických názvů registrů:

```
<symbol_declarations>     ::=  symbol <symbol_declaration> {<symbol_declaration>}
<symbol_declaration>      ::=  <symbol_name_def> = <symbolregister>
                                                        definice symbolického názvu registru
                               <symbol_name_def>                                        ::=
                                                                                  <identifier>
                                                      místo definice symbolického názvu registru
<symbol_name>             ::=  <identifier>             místo použití symbolického názvu registru
```

*Každé symbolické jméno registru v sobě nese informaci o typu, který reprezentuje (bit, byte, word, integer, longint, datetime, string, real)*

**Příklad :**

```
symbol
  BV=B10;
  BV2=B10+1;
  BV3=B10-4;
  BV4=BV3[1];
  BV5=BV3[-5];

  BITFLAG=BV.1;
  BITFLAG=B10.1;
  BITFLAG=W0.13;  { ekvivalentní k B1.5 }
  BITFLAG=W0.0;   { ekvivalentní k B0.0 }

  POMDATE=D40;

  REALPROM=R3;

  NAME=S100:20    {string začínající na adr.100, max.délka 19 (na adr.100 zaznamenána aktuální délka) }

  BV5=B20;
  NAME2=S120:20;
```

## 1.8 Deklarace konfigurací:

*Všechny objekty mají své jméno (parametr NAME). Pokud není uveden paramentr NAME, je jméno objektu shodné s typem objektu (např. objekt TERM10 má jméno TERM10). Vzhledem k tomu, že není povolena definice jednoho identifikátoru 2x, v případě použití více než jednoho objektu daného typu musíme povinně (alespoň vyjma jednoho) tyto objekty pojmenovat explicitně (T1,T2 ap.)*
*Má-li objekt přístup do bloku uživatelských registrů, jsou všechny tyto registry automaticky symbolicky pojmenovány, jméno je odvozeno od jména objektu a významu registru, např. Pbus_A, PID_W, atp.*

```
<config_declarations>     ::=  configuration <config_definitions>

<config_definitions>      ::=  <config_definition> { ; <config_definition> }

<config definition>       ::=  <HWconfig_declaration> |
                               <SWconfig_declaration>
<HWconfig_declaration>    ::=  <BIN_declaration> |                  definice binárních vstupů/výstupů
                               <TERMINAL_declaration> |                          definice terminálu
                               <ANALOG_declaration> |             definice analogových vstupů/výstupů
                               <COM_declaration> |           definice komunikačního seriového kanálu
                               <CENTRONIX_declaration>*   definice komunikačního paralelního rozhraní
*není dosud implementováno

<SWconfig_declaration>    ::=  <SAVE_declaration> |     definice úschovy registrů do zálohované paměti
                               <PIDR_declaration> |                      definice real PID-regulátoru
                               <PID_declaration> |                    definice integer PID-regulátoru
                               <FILTER_declaration>* |                  definice číslicového filtru
                               <ARCHIV_declaration>       definice archivování hodnot registrů v čase
*není dosud implementováno

<NAME_parameter>          ::=  NAME = <object_name_def>          jméno příslušného HW resp. SW objektu
```

```
<object_name_def>          ::=  <identifier>                    místo definice jména objektu
<object_name>              ::=  <identifier>                     místo použití jména objektu
```

### 1.8.1 Deklarace binárních vstupů/výstupů:

*Na základě následujících deklarací jsou definovány objekty, zabezpečující čtení resp. zápis binárních HW signálů v určitém časovém režimu do příslušných uživatelských registrů.*

```
<BIN_declaration>          ::=  <IOPBUS_declaration> |                  definice IOPbus vstupů/výstupů
                                <IODIO01_declaration> |                       definice IODIO01 vstupů
                                <IODOO01_declaration> |                      definice IODOO01 výstupů
                                <IODXO01_declaration> |                   definice IODX vstupů/výstupů
                                <IOTERM10_declaration> |        definice vstupů/výstupů na desce TERM10
                                <IODTERM10_declaration>      definice binárních vstupů/výstupů na desce
                                IOTERM10

<IOPBUS_declaration>       ::=  HWOBJ = IOPBUS, <IOPBUS_parameter> { , <IOPBUS_parameter> }
<IOPBUS_parameter>         ::=  <ADR_parameter> |                                              povinný
                                <VAR_parameter> |         nepovinný, registr povinně typu byte, délka=3
                                <NAME_parameter>                jméno PBUS rozhraní (standardně IOPBUS)
                                <IOPBUS_INITVAL_parameters>      případné předefinování default hodnoty
                                <IOPBUS_MAIN_parameter> |                                    default [ ]
                                <IOPBUS_FAST10_parameter> |                                  default [ ]
<IOPBUS_INITVAL_parametrs> ::=  A = <number> |
                                B = <number> |
                                C = <number>
```

| IOPBUS VAR_parameter | | | |
|---|---|---|---|
| *č.b.* | *název bytu* | *Význam* | *default* |
| 0 | A | INA resp. OUTA | 0 |
| 1 | B | INB resp. OUTB | 0 |
| 2 | C | INC resp. OUTC | 0 |

```
<IODIO01_declaration>      ::=  HWOBJ = IODIO01, <IODIO01_parameter> { , <IODIO01_parameter> }

<IODIO01_parameter>        ::=  <ADR_parameter> |                                              povinný
                                <VAR_parameter> |         nepovinný, registr povinně typu byte, délka=4
                                <NAME_parameter>           jméno IODIO01 rozhraní (standardně IODIO01)
                                <IODIO01_INITVAL_parameters>     případné předefinování default hodnoty
                                <IODIO01_MAIN_parameter> |                                   default [ ]
                                <IODIO01_FAST10_parameter> |                                 default [ ]

<IODIO01_INITVAL_parametrs>     ::= A = <number> |
                                    B = <number> |
                                    C = <number> |
                                    D = <number>
```

| IODIO01 VAR_parameter | | | |
|---|---|---|---|
| *č.b.* | *název bytu* | *Význam* | *default* |
| 0 | A | INA | 0 |
| 1 | B | INB | 0 |
| 2 | C | INC | 0 |
| 3 | D | IND | 0 |

```
<IODOO01_declaration>      ::=  HWOBJ = IODOO01, <IODOO01_parameter> { , <IODOO01_parameter> }

<IODOO01_parameter>        ::=  <ADR_parameter> |                                              povinný
                                <VAR_parameter> |         nepovinný, registr povinně typu byte, délka=4
                                <NAME_parameter>           jméno IODOO01 rozhraní (standardně IODOO01)
                                <IODOO01_INITVAL_parameters>     případné předefinování default hodnoty
                                <IODOO01_MAIN_parameter> |                                   default [ ]
                                <IODOO01_FAST10_parameter> |                                 default [ ]

<IODOO01_INITVAL_parametrs>     ::= A = <number> |
                                    B = <number> |
```

**C =** <number> |
**D =** <number>

| IODO001 VAR_parameter | | | |
|---|---|---|---|
| *č.b.* | *název bytu* | *Význam* | *default* |
| 0 | A | OUTA | 0 |
| 1 | B | OUTB | 0 |
| 2 | C | OUTC | 0 |
| 3 | D | OUTD | 0 |

**<IODXO01_declaration>** ::= **HWOBJ = IODXO01,** <IODXO01_parameter> { , <IODXO01_parameter> }

**<IODXO01_parameter>** ::= <ADR_parameter> | *povinný*
<VAR_parameter> | *nepovinný, registr povinně typu byte, délka=4*
**<**NAME_parameter**>** *jméno IODXO01 rozhraní (standardně IODXO01)*
<IODXO01_INITVAL_parameters> *případné předefinování default hodnoty*
<IODXO01_MAIN_parameter> | *default [ ]*
<IODXO01_FAST10_parameter> | *default [ ]*

**<IODXO01_INITVAL_parametrs>** **::= A =** <number> |
**B =** <number> |
**C =** <number> |
**D =** <number>

| IODXO01 VAR_parameter | | | |
|---|---|---|---|
| *č.b.* | *název bytu* | *Význam* | *default* |
| 0 | A | INA | 0 |
| 1 | B | INB | 0 |
| 2 | C | OUTC | 0 |
| 3 | D | OUTD | 0 |

**<IOTERM10_declaration>** ::= **HWOBJ = IOTERM10,** <IOTERM10_parameter> { , <IOTERM10_parameter> }

**<IOTERM10_parameter>** ::= <ADR_parameter> | *povinný*
<VAR_parameter> | *nepovinný, registr povinně typu byte, délka=2*
**<**NAME_parameter**>** *jméno IOTERM10 rozhraní (standardně IOTERM10)*
<IOTERM10_INITVAL_parameters> *případné předefinování default hodnoty*
<IOTERM10_MAIN_parameter> | *default [ ]*
<IOTERM10_FAST10_parameter> | *default [ ]*

**<IOTERM10_INITVAL_parametrs>** **::= IN =** <number> | **OUT =** <number>

| IOTERM10 VAR_parameter | | | |
|---|---|---|---|
| *č.b.* | *název bytu* | *Význam* | *default* |
| 0 | IN | IN | 0 |
| 1 | OUT | OUT | 0 |

**<IODTERM10_declaration>** ::= **HWOBJ = IODTERM10,** <IODTERM10_parameter> { , IODTERM10_parameter }

**<IODTERM10_parameter>** ::= <ADR_parameter> | *povinný*
<VAR_parameter> | *nepovinný, registr povinně typu byte, délka=4*
**<**NAME_parameter**>** *jméno IODTERM10 rozhraní (standardně IODTERM10)*
<IODTERM10_INITVAL_parameters> *případné předefinování default hodnoty*
<IODTERM10_MAIN_parameter> | *default [ ]*

**<IODTERM10_INITVAL_parametrs>** **::= A =** <number> | **B =** <number> | **C =** <number> | **D =** <number>

| IODTERM10 VAR_parameter | | | |
|---|---|---|---|
| *č.b.* | *název bytu* | *Význam* | *default* |
| 0 | A | IN | 0 |
| 1 | B | IN | 0 |
| 2 | C | OUT | 0 |
| 3 | D | OUT | 0 |

**<ADR_parameter>** ::= **ADR =** <hexvalue> *počáteční HW adresa v prostoru KITV40*
**<VAR_parameter>** ::= **VAR =** <symbolregister> [**:** <value>]
*počáteční registr přiřazeného bloku registrů, délka tohoto bloku v bytech*
**<FAST10_Ident>** ::= **FAST** | **FAST10**
**<PBUS_MAIN_parameter>** ::= **MAIN = [** {<PBUS_port_names>} **]**
*seznam vstupů/výstupů, obsluhovaných v normálním časovém režimu*

**<PBUS_FAST10_parameter>** ::= <FAST10_Ident> **= [** {<PBUS_port_names>} **]***seznam vstupů/výstupů, obsluhovaných v režimu FAST10*

**<PBUS_port_names>** ::= **INA** | **INB** | **INC** | **OUTA** | **OUTB** | **OUTC**

*seznam jmen portů na PBUS rozhraní, A,B,C=port buď vstup nebo výstup, případně výstup s inicializační hodnotou (seznam jednoho až tří parametrů)*

**<IODIO01_MAIN_parameter>** ::= **MAIN = [** {<IODIO01_port_names>} **]**
*seznam vstupů, obsluhovaných v normálním časovém režimu*
**<IODIO01_FAST10_parameter>** ::= <FAST10_Ident> **= [** {<IODIO01_port_names>} **]**
*seznam vstupů, obsluhovaných v režimu FAST10*

**<IODIO01_port_names>** ::= **INA** | **INB** | **INC** | **IND**
*seznam jmen portů na IODIO01 rozhraní, A,B,C,D=port vždy vstup (seznam jednoho až čtyř parametrů)*

**<IODOO01_MAIN_parameter>** ::= **MAIN = [** {<IODOO01_port_names>} **]**
*seznam výstupů, obsluhovaných v normálním časovém režimu*
**<IODOO01_FAST10_parameter>** ::= <FAST10_Ident> **= [** {<IODOO01_port_names>} **]**
*seznam výstupů, obsluhovaných v režimu FAST10*

**<IODOO01_port_names>** ::= **OUTA** | **OUTB** | **OUTC** | **OUTD**
*seznam jmen portů na IODOO01 rozhraní, A,B,C,D=port vždy výstup (seznam jednoho až čtyř parametrů)*

**<IODXO01_MAIN_parameter>** ::= **MAIN = [** {<IODXO01_port_names>} **]**
*seznam vstupů/výstupů, obsluhovaných v normálním časovém režimu*
**<IODXO01_FAST10_parameter>** ::= <FAST10_Ident> **= [** {<IODXO01_port_names>} **]**
*seznam vstupů/výstupů, obsluhovaných v režimu FAST10*

**<IODXO01_port_names>** ::= **INA** | **INB** | **OUTC** | **OUTD**
*seznam jmen portů na IODXO01 rozhraní, A,B=port vždy vstup, C+D=port vždy výstup, případně výstup s inicializační hodnotou (seznam jednoho až tří parametrů)*

**<IOTERM10_MAIN_parameter>** ::= **MAIN = [** {<IOTERM10_port_names>} **]**
*seznam vstupů/výstupů, obsluhovaných v normálním časovém režimu*
**<IOTERM10_FAST10_parameter>**::= <FAST10_Ident> **= [** {<IOTERM10_port_names>} **]**
*seznam vstupů/výstupů, obsluhovaných v režimu FAST10*

**<IOTERM10_port_names>** ::= **IN** | **OUT**
*seznam jmen portů na IOTERM10 rozhraní, jeden vstup a jeden výstup, případně výstup s inicializační hodnotou (seznam jednoho až dvou parametrů)*

**Příklady**:
```
configuration
  HWOBJ=IOPBUS, ADR=$300, VAR=B7:3, MAIN=[INA, INB, OUTC], C=$FF;       port A+B vstup, portC výstup
  HWOBJ=IODIO01, ADR=$300, VAR=B17:4, MAIN=[INA, INB], FAST10=[INC, IND];
                                                  port C+D čten po 10ms, ostatní standardně
  HWOBJ=IODOO01, ADR=$300, VAR=B17:4, FAST10=[OUTA, OUTD];                  porty B a C nepoužity
  HWOBJ=IODXO01, ADR=$300, VAR=B17:4, FAST10=[INA, OUTD];                   porty B a C nepoužity
  HWOBJ=IOTERM10, NAME=IOT, ADR=$300, VAR=B27:2, MAIN=[IN, OUT];
 vstup i vystup ošetřeny v normálním cyklu, jméno objektu IOT, tj. definovány položky IOT_IN, IOT_OUT
  HWOBJ=IODTERM10, NAME=IODT, ADR=$320, VAR=B30:4, MAIN=[A, D];
vstup A a vystup D ošetřeny v normálním cyklu, jméno objektu IODT, tj. definovány položky IODT_A,
IODT_D
```

### 1.8.2 Deklarace analogových vstupů/výstupů:

*Na základě následujících deklarací jsou definovány objekty, zabezpečující čtení resp. generování analogových signálů pomocí desek AD a DA převodníků v určitém časovém režimu do příslušných uživatelských registrů.*

<**ANALOG_declaration**> ::= <ADDA_declaration> | *definice vstupů/výstupů desky ADDA*
<IOATERM10_declaration>*definice analogových vstupů/výstupů na desce IOTERM10*

**<ADDA_declaration>** ::= **HWOBJ = ADDA,** <ADDA_parameter> | <ADDA_declaration> , <ADDA_parameter>
**<ADDA_parameter>** ::= <ADR_parameter> | *povinný*
<VAR_parameter> | *povinný, registr typu integer, délka=20*
**<**NAME_parameter**>** *jméno ADDA rozhraní (standardně ADDA)*
<ADDA_MAIN_parameter> | *default [ ]*
<ADDA_INITVAL_parameters>

| **ADDA VAR_parameter** | | | |
|---|---|---|---|
| *č.b.* | *název bytu* | *Význam* | *default* |
| 0+1 | IN0 | IN0 | 0 |
| 2+3 | IN1 | IN1 | 0 |
| 4+5 | IN2 | IN2 | 0 |
| 6+7 | IN3 | IN3 | 0 |
| 8+9 | IN4 | IN4 | 0 |
| 10+11 | IN5 | IN5 | 0 |
| 12+13 | IN6 | IN6 | 0 |
| 14+15 | IN7 | IN7 | 0 |
| 16+17 | OUT0 | OUT0 | 0 |
| 18+19 | OUT1 | OUT1 | 0 |
| 20+21 | OUT2 | OUT2 | 0 |
| 22+23 | OUT3 | OUT3 | 0 |
| 24+25 | OUT4 | OUT4 | 0 |
| 26+27 | OUT5 | OUT5 | 0 |

Pokud jsou použity výstupy OUT4 a OUT5, není na desce IOADDA přístupný ani jeden ze vstupních kanálnů IN0 až IN7. Pokud deklarace hw objektu obsahuje obojí překladač hlásí chybu.

**<ADDA_MAIN_parameter>** ::= **MAIN = [** {<ADDA_port_name>} **]**
*seznam vstupů/výstupů, obsluhovaných v normálním časovém režimu*

**<ADDA_FAST_parameter>** ::= <FAST10_Ident> **= [** {<ADDA_port_name>} **]**
(zatím implementovány pouze vstupy)

**<ADDA_port_name>** ::= **IN0=<**ADDA_INtype> | **IN1=<**ADDA_INtype> | **IN2=<**ADDA_INtype> |
**IN3=<**ADDA_INtype> | **IN4=<**ADDA_INtype> | **IN5=<**ADDA_INtype> |
**IN6=<**ADDA_INtype> | **IN7=<**ADDA_INtype> |
**OUT0** | **OUT1** | **OUT2** | **OUT3** | **OUT4** | **OUT5**
*seznam použitých analog. vstupů/výstupů. Je-li vstup IN0 resp. IN2 resp. IN4 resp. IN6 nadefinován jako diferenciální, nesmí být uveden v definici vstup IN1 resp. IN3 resp. IN5 resp. IN7. Tyto vstupy samy nesmějí být nadefinovány jako diferenciální.*
*Analogové vstupy 0 až 7 a analogové výstupy 4 a 5 nesmí být použity zároveň (limitováno hardwarem).*

**<ADDA_INtype>** ::= **SU** | *singulární analog.vstup s digit.výstupem v rozsahu 0..4095*
**SB** | *singulární analog.vstup s digit.výstupem v rozsahu -4095..4095*
**DU** | *diferenciální analog.vstup s digit.výstupem v rozsahu 0..4095*
**DB** *diferenciální analog.vstup s digit.výstupem v rozsahu -4095..4095*

**<ADDA_INITVAL_parametrs>** **::= OUT0 =** <number> | **OUT1 =** <number>

**<IOATERM10_declaration>** ::= **HWOBJ = IOATERM10,** <IOATERM10_parameter> |
<IOATERM10_declaration> , <IOATERM10_parameter>
**<IOATERM10_parameter>** ::= <ADR_parameter> | *povinný*
<VAR_parameter> | *nepovinný, registr povinně typu byte, délka=62*
**<**NAME_parameter**>** *jméno IOATERM10 rozhraní (standardně IOATERM10)*
<IOATERM10_INITVAL_parameters> *případné předefinování default hodnoty*
<IOATERM10_CASE_parameter> | *default [ ]*

| **IOATERM10 VAR_parameter** | | | |
|---|---|---|---|
| *č.b.* | *název bytu* | *Význam* | *default* |
| 0+1 | IN0 | IN0 | 0 |
| 2+3 | IN1 | IN1 | 0 |
| 4+5 | IN2 | IN2 | 0 |
| 6+7 | IN3 | IN3 | 0 |
| 8+9 | IN4 | IN4 | 0 |
| 10+11 | IN5 | IN5 | 0 |
| 12+13 | IN6 | IN6 | 0 |
| 14+15 | IN7 | IN7 | 0 |
| 16+17 | IN8 | IN8 | 0 |
| 18+19 | IN9 | IN9 | 0 |

| | | | |
|---|---|---|---|
| 20+21 | IN10 | IN10 | 0 |
| 22+23 | IN11 | IN11 | 0 |
| 24+25 | IN12 | IN12 | 0 |
| 26+27 | IN13 | IN13 | 0 |
| 28+29 | IN14 | IN14 | 0 |
| 30+31 | IN15 | IN15 | 0 |
| 32+33 | OUT0 | OUT0 | 0 |
| 34+35 | OUT1 | OUT1 | 0 |
| 36+37 | OUT2 | OUT2 | 0 |
| 38+39 | OUT3 | OUT3 | 0 |
| 40+41 | OUT4 | OUT4 | 0 |
| 42+43 | OUT5 | OUT5 | 0 |
| 44+45 | INT | INT | 0 |

<IOATERM10_CASE_parameter> ::= **CASE = [** <IOATERM10_port_names> **]** | **CASE = [ ]**
*seznam vstupů/výstupů, obsluhovaných v csačasovém režimu*

<IOATERM10_port_names> ::= **IN0**| .. | **IN15** | **OUT0** | .. | **OUT5** | **INT** | <IOATERM10_port_names> **=** <IOATERM10_port_type> |
<IOATERM10_port_names>, <IOATERM10_port_names>
*seznam použitých analog. vstupů/výstupů.*

<IOATERM10_INITVAL_parametrs> ::= OUT0 = <number> |..| OUT5 = <number>

<IOATERM10_port_type> ::= **BIT8** | **BIT10** | **BIT12** | **BIT14** | **BIT16** | **PT100**

**Příklady:**
```
configuration
  HWOBJ=ADDA, ADR=$300, VAR=I100:20, NAME=ADDA01, MAIN=[IN0=DU,IN2=DB,IN4=SU,IN5=SU,OUT0=0,OUT1=0];
  HWOBJ=ADDA, ADR=$320, VAR=I200:20, NAME=ADDA02, MAIN=[IN0=DB, IN2=DB, IN4=DB];
  HWOBJ=IOATERM10, ADR=$330, VAR=I300:46, NAME=ATERM, CASE=[IN0,IN2=Bit12, Out0,Out1];
{ vstupy na první desce převodníků 0 a 2 diferenciální, vstupy 4 a 5 singulární, vstupy 6 a 7
nepoužity, na druhé desce převodníků použity pouze 3 diferenciální vstupy 0, 1 a 2 }
```

### 1.8.3 Deklarace komunikačního seriového rozhraní:

*Na základě následujících deklarací jsou definovány objekty, zabezpečující komunikaci po sériovém kanálu mezi druhým zařízením a sestavou KIT, přičemž obsah přenášených zpráv se tvoří na základě obsahu příslušných uživatelských registrů resp. dekóduje do příslušných uživatelských registrů. Vyslání a přijetí zprávy a stavy bytů objektu COM umožňují procedury pro komunikaci.*

**<COM_declaration>** ::= **HWOBJ = COM** <COM_parameters>

**<COM_parameters>** ::= **NAME <**NAME_parameter**>** | *jméno objektu, standardně COM*
**VAR** <VAR_parameter> | *nepovinný, registr povinně typu byte, délka=*
<PAR_parametr> | *povinný, inicializace COMu*

**<PAR_parametr>** ::= **PAR =** <string_const>

| **COM VAR_parameter** | | | |
|---|---|---|---|
| *č.b.* | *název bytu* | *Význam* | *Default* |
| 0 | CTRL | řídící byte | 0 |
| 0.0 | OFF | komunikace vypnuta | [ ] |
| 0.1 | ON | komunikace zapnuta | 0 |
| 0.2 | ERR | chyba komunikace | 0 |
| 0.4 | SEND | odesílá se | 0 |
| 0.5 | SENDERR | chyba odesílání | 0 |
| 0.6 | REC | přijímá se | 0 |
| 0.7 | RECERR | chyba přijímání | 0 |
| 1 | DNODE | node (odesílatele) | 0 |
| 2 | RSNODE | node (odesílatele) | 0 |

**Příklady:**
```
configuration
  HWOBJ=COM, NAME=COM, VAR=Byte, PAR= "NAM=COM COM=2    IRQ=3 BD=9600 BIT=8 STO=2 PAR=N LRB=10001"
```

Komunikační objekt COM se dá řídit také standardními komunikačními procedurami <com_procedures>.

### 1.8.4 Deklarace komunikačního paralelního rozhraní:

*Na základě následujících deklarací jsou definovány objekty, zabezpečující komunikaci po paralelním rozhraní Centronix.*

```
<CENTRONIX_declaration>    ::=   HWOBJ = CENTRONIX <CENTRONIX_parameters>

<CENTRONIX_parameters>     ::=   NAME <NAME_parameter> |
                                 ADR <number>
```

```
Příklady:
configuration
  HWOBJ=CENTRONIX, NAME=PRINTER, ADR=$300;
```

bude specifikováno dodatečně !

### 1.8.5 Deklarace úschovy registrů do zálohované paměti:

*Na základě následujících deklarací jsou definovány objekty, zabezpečující úschovu množiny uživatelských registrů do zálohované paměti. Tyto uživatelské registry tak mají nezměněný obsah po SW Resetu i výpadku napájení.*

```
<SAVE_declaration>         ::=   SWOBJ = SAVER, <SAVE_parameter>
<SAVE_parameter>           ::=   SAVE = [ <Registers> ]                    definice seznamu registrů
<Registers>                ::=   <Register> |                             jméno libovolného registru
                                 <Register>..<Register> |      jména dvou registrů, tvořících interval
                                                    (absolutní hodnota adresy 1. registru musí být menší)
                                 <Registers> , <Registers>
```

```
Příklady:
configuration
  SWOBJ=SAVER, SAVE=[B10..B90,R10..R20];
```

### 1.8.6 Deklarace terminálů (operátorský vstup/výstup):

```
<TERMINAL_declaration>     ::=   <TERM10_declaration> |          definice oper.vstupů/výstupů na TERM10
                                 <TERM01_declaration> |          definice oper.vstupů/výstupů na TERM01
                                 <TERM03_declaration> |          definice oper.vstupů/výstupů na TERM03
                                 <GENSTR_declaration> |    definice oper.vstupů/výstupů do reg. bufferu
                                 <TERMP_declaration> |         definice oper.výstupů na paral. rozhraní
                                 <TERMS_declaration>         definice oper.vstupů/výstupů na ser. kanál

<TERM10_declaration>       ::=   HWOBJ = TERM10, <TERM10_parameter> {, <TERM10_parameter }

<TERM10_parameter>         ::=   <ADR_parameter> |             povinný, je-li terminál součástí sestavy
                                 <VAR_parameter> |        nepovinný, registr povinně typu byte, délka=5
                                 <NAME_parameter>            jméno TERM10 rozhraní (standardně TERM10)
                                 <TERM10_INITVAL_parameters>     případné předefinování default hodnoty
<TERM10_INITVAL_parametrs> ::=   SCRNO = <number> |
                                 BEEP = <number> |
                                 LED = <number>
```

| TERM10 VAR_parameter | | | |
|---|---|---|---|
| *č.b.* | *název bytu* | *Význam* | *Default* |
| 0 | SCRNO | číslo obrazovky | 0 |
| 1 | LEED | LED-diody | 0 |
| 2 | CTRL | řídici bajt | 0 |
| 2.0 | BEEP | bzučák | 0 |
| 2.1 | START | stisknuté tl.START | 0 |
| 2.2 | STOP | stisknuté tl.STOP | 0 |
| 2.3 | REFRESH | zadost o refresh | 0 |

*pozn. Bit Refresh je normálně v nule. Nasatvením tohoto bitu do jedničky žádá uživatelský program o překreslení displeje. Po překreslení je bit automaticky shozen do nuly. (Pozor: nastavení bitu do jedničky a následné čekání na jeho spadnutí do nuly způsobí deadlock )*

**<TERM01_declaration>** ::= **HWOBJ = TERM01,** <TERM01_parameter> { , <TERM01_parameter }

**<TERM01_parameter>** ::= <VAR_parameter> | povinný, registr povinně typu byte, délka=2
**<**NAME_parameter**>** | jméno TERM01 rozhraní (standardně TERM01)
<ADR_parameter> adresa portu sériového kanálu
<IRQ_parameter> číslo přerušení od sériového kanálu

<TERM01_INITVAL_parameters> případné předefinování default hodnoty
**<TERM01_INITVAL_parametrs>** ::= **SCRNO =** <number>

| **TERM01 VAR_parameter** | | | |
|---|---|---|---|
| *č.b.* | *název bytu* | *Význam* | *default* |
| 0 | SCRNO | číslo obrazovky | 0 |

*Terminalova sekce programu pro Term01 může obsahovat pouze příkazy print, position, edit, wait, case, onkey, editenter a help. Ostatní přikazy nebudou rozeznány. Není také možný výpis binárního 16-ti bitového čísla (typu word).*

<TERM03_declaration> ::= HWOBJ = TERM03, <TERM03_parameter> { , <TERM03_parameter }

<TERM03_parameter> ::= <VAR_parameter> | *povinný, registr povinně typu byte, délka=2*
<NAME_parameter> | *jméno TERM03 rozhraní (standardně TERM03)*
<COMNAME_parameter> |*jméno dříve nadefinovaného přenos. kanálu typu COM*
<TERM03_INITVAL_parameters> *případné předefinování default hodnoty*
<TERM03_INITVAL_parametrs> ::= SCRNO = <number> |
BEEP = <number>

| **TERM03 VAR_parameter** | | | |
|---|---|---|---|
| *č.b.* | *název bytu* | *Význam* | *default* |
| 0 | SCRNO | číslo obrazovky | 0 |
| 1 | BEEP | Beeper | 0 |

bude specifikováno dodatečně !

**<GENSTR_declaration>** ::= **SWOBJ = GENSTR,** <GENSTR_parameter> { , <GENSTR_parameter> }

**<GENSTR_parameter>** ::= **<**NAME_parameter**>** | *jméno GENSTR rozhraní (standardně GENSTR)*
<VAR_parameter> | nepovinný, *registr povinně typu byte*
<BUFF_parameter> *povinná proměnná výstupního bufferu generátoru*

| **GENSTR VAR_parameter** | | | |
|---|---|---|---|
| *č.b.* | *název bytu* | *Význam* | *default* |
| 0 | SCRNO | číslo obrazovky | 0 |

**<TERMP_declaration>** ::= **HWOBJ = TERMP,** <TERMP_parameter> { , TERMP_parameter }

**<TERMP_parameter>** ::= **<**NAME_parameter**>** | *jméno TERMP rozhraní (standardně TERMP)*
<CENTRONIXNAME_parameter> |*jméno dříve nadefinovaného přenos. kanálu typu CENTRONIX*
<TERM03_INITVAL_parameters> *případné předefinování default hodnoty*

bude specifikováno dodatečně !

**<TERMS_declaration>** ::= **HWOBJ = TERMS,** <TERMS_parameter> { , TERMS_parameter }

**<TERMS_parameter>** ::= **<**NAME_parameter**>** | *jméno TERMP rozhraní (standardně TERMP)*
<COMNAME_parameter> |*jméno dříve nadefinovaného přenos. kanálu typu COM*

bude specifikováno dodatečně !

**Příklady:**

```
configuration
  HWOBJ=TERM10, NAME=T10, ADR=$300, VAR=B100:5;  { terminal v sestave }
  HWOBJ=TERM10, NAME=T10, COMNAME=COM1, VAR=B100:5; { terminal pripojen pres seriovy kanal }*
  HWOBJ=GENSTR, VAR=Byte, BUFF=S0; { terminal generujici stringy }
  HWOBJ=TERM01, NAME=T1, ADR=$2320, IRQ=3; {Term01 přes sériový kanál}
  HWOBJ=TERM03, NAME=T3, COMNAME=COM2, VAR=B120:2; *
```

*není dosud implementováno

### 1.8.7 Deklarace PIDR-regulátoru:

*Na základě následujících deklarací jsou definovány objekty, pracující jako autonomní real PID-regulátory.*

```
<PIDR_declaration>        ::=  SWOBJ = PIDR, <PIDR_parameter> { , <PIDR_parameter> }

<PIDR_parameter>          ::=  <VAR_parameter> |          povinný, registr povinně typu REAL, délka=16
                               <NAME_parameter> |              jméno PIDR regulátoru (standardně PIDR)
                               <PIDR_TS_parameter> |                  zadání periody výpočtu [v sec]
                               <PIDR_INITVAL_parameters>       případné předefinování default hodnoty

<PIDR_TS_parameter>       ::=  TS <number> |                                      perioda vzorkování
<PIDR_INITVAL_parametrs>  ::=  AUTO = <number> |                  0=manuální režim, jinak automat
                               W = <float_number> |                             požadovaná hodnota
                               Y = < float_number> |                              naměřená hodnota
                               U = < float_number> |      akční zásah (v případě AUTO=0 se rovná UMAN)
                               UMAN = < float_number> |                  manuální akční zásah
                               KP = < float_number> |               proporcionální konstanta
                               KI = < float_number> |                   integrační konstanta
                               KD = < float_number> |                    derivační konstanta
                               RELK = < float_number> |                   násobící konstanta
                               MINU = < float_number> |                minimální akční zásah
                               MAXU = < float_number> |                maximální akční zásah
                               MINY = < float_number> |             minimální naměřená hodnota
                               MAXY = < float_number> |             maximální naměřená hodnota
                               EGAP = < float_number> |                     zapínání regulátoru
                               IGAP = < float_number> |           zapínání integrační složky
                               DGAP = < float_number>              zapínání derivační složky
```

| PIDR VAR_parameter | | | |
|---|---|---|---|
| *č.b.* | *název bytu* | *Význam* | *Default* |
| 0 | AUTO | zap./vyp. auto regulace | 0 |
| 1 | W | požadovaná hodnota | 0 |
| 2 | Y | naměřená hodnota | 0 |
| 3 | U | spočtená výst. hodnota | 0 |
| 4 | Uman | manuální výst. hodnota | 0 |
| 5 | KP | proporcionální konstanta | 0 |
| 6 | KI | integrační konstanta | 0 |
| 7 | KD | derivační konstanta | 0 |
| 8 | RELK | násobící konstanta | 1 |
| 9 | MINU | min. výstupní hodnota | 0 |
| 10 | MAXU | max. výstupní hodnota | 100 |
| 11 | MINY | min. naměřená hodnota | 0 |
| 12 | MAXY | max. naměřená hodnota | 100 |
| 13 | EGAP | zapínání regulátoru | 0% |
| 14 | IGAP | zapínání I složky | 100% |
| 15 | DGAP | zapínání D složky | 100% |

```
Příklady:
configuration
  SWOBJ=PIDR, VAR=R0:16, TS=2;
  SWOBJ=PIDR, NAME=REGULATOR, VAR=R100:16, TS=2, AUTO=1, W=55, KP=10, KI=3;
```

### 1.8.8 Deklarace Integerového PID-regulátoru:

Na základě následujících deklarací jsou definovány objekty, pracující jako autonomní integer PID-regulátory.

```
<PIDI_declaration>        ::=  SWOBJ = PID, <PIDI_parameter> { , <PIDI_parameter }

<PIDI_parameter>          ::=  <VAR_parameter> |       povinný, registr povinně typu integer, délka=28
                               <NAME_parameter> |              jméno PID regulátoru (standardně PID)
                               <PIDI_TS_parameter> |                  zadání periody výpočtu [v sec]
                               <PIDI_INITVAL_parameters>       případné předefinování default hodnoty
<PIDI_TS_parameter>       ::=  TS <number> |                                      perioda vzorkování
```

```
<PIDI_INITVAL_parametrs>  ::=  AUTO = 0 | AUTO = 1                  0=manuální režim, jinak automat
                               W = <sign_number> |                        požadovaná hodnota
                               Y = <Sign_number> |                          naměřená hodnota
                               U = <Sign_number> |       akční zásah (v případě AUTO=0 se rovná UMAN)
                               UMAN = <Sign_number> |                  manuální akční zásah
                               KP = <Sign_number> |                 proporcionální konstanta
                               KI = <Sign_number> |                     integrační konstanta
                               KD = <Sign_number> |                      derivační konstanta
                               RELK = <Sign_number> |                    násobící konstanta
                               MINU = <Sign_number> |                 minimální akční zásah
                               MAXU = <Sign_number> |                 maximální akční zásah
                               MINY = <Sign_number> |              minimální naměřená hodnota
                               MAXY = <Sign_number> |              maximální naměřená hodnota
                               EGAP = <Sign_number> |                       zapínání regulátoru
                               IGAP = <Sign_number> |              zapínání integrační složky
                               DGAP = <Sign_number>                zapínání derivační složky
```

*Ostatní podrobnosti - viz deklarace PID-regulátoru.*

```
Příklady:
configuration
  SWOBJ=PIDI, VAR=I0:28, TS=2;
  SWOBJ=PIDI, NAME=REGULATOR, VAR=I100:28, TS=2, AUTO=1, W=55, KP=10, KI=3;
```

### 1.8.9 Deklarace číslicových filtrů:

*Na základě následujících deklarací jsou definovány objekty, které zabezpečují funkci číslicových filtrů. Každý typ filtru má svůj algoritmus filtrování:*

| ***Přehled filtrů*** | | | |
|---|---|---|---|
| *SWOBJ* | *množina registrů* | *Algoritmus* | |
| *FILTER0* | *Ra,Rb* | *Rb := (9*Ra+Rb)/10* | |
| | | | |

```
<FILTER_declaration>      ::=  SWOBJ = FILTERx, <FILTER_parameter> {, <FILTER_parameter> }

<FILTER_parameter>        ::=  <NAME_parameter> |                    jméno filtru (standardně FILTERx)
                               <FILTER_MAIN_parameter> |
                                        množina registrů, filtrace po každém průchodu procedurou MAIN
                               <VAR_parameter>                        sada vnitřních proměnných filtru
                               <FILTER_INITVAL_parameters>     případné předefinování default hodnoty


<FILTER_MAIN_parameter>   ::=  MAIN = [ <registers> ]
```

bude specifikováno dodatečně !

```
Příklady:
configuration
*  SWOBJ=FILTER1, MAIN=[I10, I20];
```

**není dosud implementováno*

### 1.8.10 Deklarace HW-objektu IOFLEXPOS:

*Na základě následujících deklarací je definován objekty, obsluhující desku IOFLEX01 s programem pro odměřování polohy.*

```
<IOFLEXPOS_declaration>   ::=  HWOBJ = IOFLEXPOS, <IOFLEXPOS_parameter> { , <IOFLEXPOS_parameter> }

<IOFLEXPOS_parameter>     ::=  <VAR_parameter> |          nepovinný, registr typu lingint, délka=28
                               <NAME_parameter> |              jméno objektu (standardně IOFLEXPOS)
                               <ADR_parameter> |                 baze desky IOFLEX01 v IO prostoru
                               FAST                            definovani obsluhy desky ve fast10
```

| **IOFLEXPOS VAR_parameter** |
|---|

| *č.b.* | *název položky* | *Význam* | *Default* |
|---|---|---|---|
| 0 | TIME | čas od poslední ho měření v µs | 0 |
| 4 | POS0 | hodnota čidla 0 | 0 |
| 8 | POS1 | hodnota čidla 1 | 0 |
| 12 | POS2 | hodnota čidla 2 | 0 |
| 16 | POS3 | hodnota čidla 3 | 0 |
| 20 | STATUS0 | stav čidla 0 | 0 |
| 21 | STATUS1 | stav čidla 1 | 0 |
| 22 | STATUS2 | stav čidla 2 | 0 |
| 23 | STATUS3 | stav čidla 3 | 0 |
| 24 | CMD0 | příkaz pro čidlo 0 | 0 |
| 25 | CMD1 | příkaz pro čidlo 1 | 0 |
| 26 | CMD2 | příkaz pro čidlo 2 | 0 |
| 27 | CMD3 | příkaz pro čidlo 3 | 0 |
| 28 | APPLYCMD | potvrzení příkazu | 0 |

*Položky STATUS0-3 mohou nabývat pouze dvou hodnot 0 a 1. 0 v případě že čidlo ještě nebylo zkalibrováno a 1 v opačném v případě.*
*Položky CMD0-3 jsou příkazy pro jednotlivá čidla. 0 – bez příkazu, 1 – okamžité nulování (kalibrace), 2 – nulování při přechodu přes nulu, 3 – nulování při přechodu přes nulu za splnění vnější podmínky.*

```
Příklady:
configuration
  HWOBJ=IOFLEXPOS, ADR=$2310;
  HWOBJ=IOFLEXPOS, NAME=FLEX, VAR=L100:28, ADR=$2310, FAST;
```

### 1.8.11 Deklarace Tabulky:

*Objekt tabulka umožňuje definovat dvě pole vektorů, x a y a třídit vektory y podle hodnot x.*

**<TAB_declaration>** ::= **SWOBJ = TAB,** <TAB_parameter> {, <TAB_parameter> }

**<TAB_parameter>** ::= **<**NAME_parameter**>** | *jméno generatoru (standardně TAB)*
<VAR_parameter> | *počet platných položek v tabulce*
<X_parameter> | *pole prvků libovolného typu - vektor X*
<Y_parameter> *pole prvků libovolného typu - vektor Y*

<**X_parameter**> ::= **X =** <symbol_register>
<**Y_parameter**> ::= **Y =** <symbol_register>

### 1.8.12 Deklarace archivování průběhu hodnot:

*Na základě následujících deklarací jsou definovány objekty, zabezpečující úschovu množiny uživatelských registrů na virtuální disk. Ukládání se provádí buď automaticky s danou periodou (použití pro měřené hodnoty) nebo na požádání vyvoláním procedury* ***ARCHIVSAVE(name)*** *(použití jako reakce na zadání hodnoty od operátora nebo vzniklý Alarm) do zálohované paměti. Každý archiv má při své definici určen maximální povolený počet položek, při jejich překročení se některé údaje ztrácejí. V případě archivu typu* ***ARCHIVLAST*** *se do kruhového bufferu ukládá posledních Size záznamů, nejstarší záznamy při překročení limitu maximálního počtu záznamů ruší. V případě archivu typu* ***ARCHIVFIRST*** *se do bufferu uloží maximálně Size záznamů, zápis dalších záznamů po naplnění bufferu se již neprovádí. Dále je nadefinován speciální archiv* ***ARCHIVERROR****, do kterého se automaticky zapisují chyby vzniklé za běhu programu. Uživatel si může pomocí tohoto archivu vypisovat vzniklé chyby na obrazovku terminálu tak, že nadefinuje u tohoto archivu LOAD registry (k SAVE registrům nemá přístup) o celkové délce 11 Bytů, (význam uveden dále). POZOR !* ***ARCHIVERROR*** *je možné nadefinovat pouze jednou !*

*Hodnoty v archivu lze mazat pomocí procedur* ***ARCHIVCLEARALL(name)*** *a* ***ARCHIVCLEARLAST(name)***

*Hodnoty z archivu lze číst procedurami* ***ARCHIVLOAD(name,errfl)*** *(čtou se postupně všechny hodnoty popořadě, je-li ErrFl<>0, pak další záznam neexistuje), čtení vybrané hodnoty můžeme nastavit procedurami* ***ARCHIVSEEKFIRST(name)*** *(přechod na první záznam),* ***ARCHIVSEEKLAST(name)*** *(přechod na poslední záznam) a* ***ARCHIVSEEK(name)*** *(nalezne se záznam s první položkou větší nebo rovnou hodnotě, přítomné v registru, uvedeném v parametru* ***SEEK****)*

*Počet záznamů v archivu zjistíme procedurou* ***ARCHIVCOUNT(name,result).***

```
<ARCHIV_declaration>      ::=  SWOBJ = ARCHIVFIRST, <ARCHIV_parameter>                    |
                                                    archiv, uchovávající prvních Size záznamů
                               SWOBJ = ARCHIVLAST, <ARCHIV_parameter>
                                                  archiv, uchovávající posledních Size záznamů
                               SWOBJ = ARCHIVERROR,  <ARCHIVErr_parameter>
                                                   archiv error je nadefinovan jako archivlast
                               <ARCHIV_parameter><ARCHIV_declaration> ,
<ARCHIV_parameter>        ::=  NAME = <ident_definition> |                          jméno archivu
                               SIZE = <number> |                  maximální počet záznamů v archivu
                               PER = <number> |                  perioda automatického ukládání [s]
                                 (není-li uvedena resp. je-li =0, pak ukladaní pouze pomocí procedury)
                               FORMAT = [ <String>,<String> | ,<String>,<String> | .. ]
                                seznam stringů, které se předávají po komunikaci jako hlavicky archivů
                                     (není-li uvedena automatickly se vygeneruje podle názvů registrů)
                               SAVE = [ <Registers> ] |       seznam registrů, které se mají uchovávat
                               LOAD = [ <Registers> ]
                               seznam registrů, do kterých se mají číst nahrané hodnoty (musí
                               odpovídat počtem Bytů seznamu uvedenému v parametru SAVE)

<ARCHIVErr_parameter>     ::=  NAME = <ident_definition> |                          jméno archivu
                               LOAD = [ <Registers> ]
                               seznam registrů, do kterých se mají číst nahrané hodnoty v tomto
                               složení:
                               ErrTime=Longint,ErrPos=Byte,ErrCode=Byte,Segment=word,Offset=Word,
                               celkem 11 bytů
```

```
Příklady:
constant
  ErrTime=L110; ErrPos=B10; ErrPos=B11; ErrCode=B12; ErrSeg=W14; ErrOff=W16;
configuration
  HWOBJ=PBus, ADR=$300:3, MAIN=[INA,INB,OUTC];
  SWOBJ=ArchivError, NAME=ERRORS, LOAD=[ErrTime,ErrVer,ErrPos,ErrCode,ErrSeg,ErrOff];
                                    archiv pro záznam RUN=time chyb, možnost výpisu na obrazovku
                                                                        pomocí Err registrů.
  SWOBJ=ArchivLast, SAVE=[Pbus_A,Pbus_B], LOAD=[B124..B125], SIZE=1000, PER=300;
                                archiv pro periodický záznam průběhu vstupních hodnot na Pbus desce,
                                        vstupy A a B (perioda 300sec), posledních 1000 záznamů,
                                    možnost výpisu na obrazovku pomocí registrů B124,125 a dalších.

  SWOBJ=ArchivLast, NAME=DRUHY, SAVE=[B126,Pbus_B,PBus_C], LOAD=[W33,B127], SIZE=500, PER=30;
                          další archiv pro periodický záznam průběhu vstupních hodnot na Pbus desce.
  SWOBJ=archivelast, NAME=TRETI, SIZE=8, PER=1, SAVE=[C_PACK,R_Y], LOAD=[LoadTime,LoadY],
FORMAT=["ČAS","%t","Akční Zásah","%f"](jeden z archivů musí být přejmenován, jinak hlásí překladač
chybu 2 stejných jmen objektů)
```

# 1.9 Deklarace terminálů:

*V této deklaraci se definuje vždy 1 obrazovka (formát výpisu) pro konkrétní terminál.*

## 1.9.1 Deklarace obrazovky terminálu

```
<terminal_declaration>    ::=  <terminal_heading> <terminal_screen_body>
<terminal_heading>        ::=  terminal <terminal_object_name> : <screenNo> ;
<terminal_object_name>    ::=  <object_name>       jméno terminálu, nadefinované v sekci configuration
<screenNo>                ::=  <const_expression>   číslo obrazovky terminálu, která se bude definovat

<terminal_screen_body>    ::=  begin <screen_members> [help <help_members>] end ;
<screen_members>          ::=  <screen_member> { < screen_member> }

<screen_member>           ::=  <screen_units> |                    popisy obrazovky, které se opakují
                               <screen_statement>         seznam příkazů pro odezvu terminálu na vstup

<screen_units>            ::=  <screen_descriptions> |
                               begin <screen_descriptions> end; |
                                                  seznam popisů pro výpis na obrazovku a seznam příkazů
```

```
<help_members>            ::=  <screen_descriptions>
                                                  seznam popisů pro výpis na help obrazovku

<screen_descriptions>     ::=  <screen_description> { < screen_desctiption }

<screen_description>      ::=  <case_description> |      variantní popis dle hodnoty určitého registru
                               <graph_members>                                    grafické prvky

<graph_members>           ::=  <graph_member>; { <graph_members> ; }

<graph_member>            ::=  <bitmap_description> |                popis použité podkladové bitmapy
                               <font_description> |                          popis použitého fontu
                               <position_description> |      definice pozice, od které se má vypisovat
                               <print_description> |            popis vypisovaného textu a proměnných
                               <point_description> |                                    výpis bodu
                               <line_description> |                                   výpis přímky
                               <circle_description> |                               výpis kružnice
                               <rect_description> |                                 výpis obdélníku
                               <fill_description> |                       výpis vyplněného obdélníku
                               <graph_description> |                                 výpis grafu
                               <graphxy_description> |                 výpis grafu podle x a y složek
                               <bar_description>                                  výpis bargrafu
                               begin <graph_members> end

<edit_description> se smí na jedné obrazovce vyskytnout max. 1x !

<case_description>        ::=  case <regBV> of <case_variant_descriptions>
                                [else <graph_members>] end;
                                                    variantní výpis dle hodnoty registru

<case_variant_descriptions>::= <case_variant_description> { <case_variant_description> }

<case_variant_description> ::= <case_constants> : <graph_member>;
                                                 prvek case sekce, jedna varianta výpisu

<case_constants>          ::=  <case_constant> { , <case_constant> }          blok konstant case sekce

<case_constant>           ::=  <const_expression> |
                               <const_expression>..<const_expression>
                                          konstanta, nebo rozsah, ve kterém má být hodnota registru

<screen_statement>        ::=  <onkey_statements> |                 příkaz prováděný po stisku klávesy
                               <editenter_description> |       příkazy vykonávané po úspěšném ukončení
                                                                                        editace
                               <edit_description> |              popis vypisované a editované proměnné
                               <editp_description> |                              popis editace hesla
                               <wait_statement>
                                     změna obrazovky prováděná po uplynutí timeoutu bez stisku klávesy
```

### 1.9.2 Definice výpisů na obrazovku

```
<bitmap_description>      ::=  bitmap <bitmap_number>               výpis bitmapy daného čísla 0 až 7
<bitmap_number>           ::=  <RegnumC_Index>

Číslo bitmapy má smysl pouze pro terminály TERM10 a TERM03.
Číslo bitmapy můžeme zadat v rozmezí 0..8 resp. 255:
   255                    = žádná bitmapa, (default hodnota)
   0                      = implicitní bitmapa SofCon,
   1..8                   = uživatelská bitmapa, nahrávaná z PC

<font_description>        ::=  font <font_number>  následující výpisy pomocí fontu daného čísla 0 až 7
<font_number>             ::=  <RegnumC_Index >

Číslo fontu má smysl pouze pro terminály TERM10 a TERM03.
Číslo fontu můžeme zadat v rozmezí 1..8
                               1..4                  = standardní font z EPROM (default hodnota = 1)
                               5..8                        = uživatelský font, nahrávaný z PC

<position_description>    ::=  position <px>,<py>                 následující výpisy od zadané pozice
<px>                      ::=  <RegnumC_Index>
<py>                      ::=  <RegnumC_Index>
```

```
Povolený rozsah pozice x a y je závislý na druhu terminálu:
   TERM10   x=0..239     y=0..127   default hodnota 0,0
   TERM03   x=0..127     y=0..63    default hodnota 0,0
   TERM01   x=0..15      y=0..3     default hodnota 0,0
```

*Default hodnoty pro **bitmap**, **font** a **position** platí od počátku definice obrazovky až do jejich explicitní změny.*
*Nově definovaná hodnota **font** platí až do následující změny.*
*Jednotlivé textové výpisy popisem **print** se provádějí za sebou až do další explicitní změny **position**.*

**<print_description>** ::= **print** <print_elements> *výpis zadaného textu*

**<print_elements>** ::= <print_element> { , <print_element> }

**<print_element>** ::= <string_const> | *výpis zadaného textu*
<base><regVBWIL>:<chlen>| *popis vypisované celočíselné proměnné*
<base><regVBWIL>:<chlen>:<frlen>| *popis vypisované celočíselné proměnné*
<base><regVBWIL> | *popis vypisované celočíselné proměnné*
<regR>:<chlen>:<frlen>| *popis vypisované reálné proměnné*

<regR> | *popis vypisované reálné proměnné*
<regS>:<chlen> | *popis vypisované proměnné typu string*
**date** <regW> | popis výpisu datumu dle 8-bytového pole (unpack format)
**date** <regL> | popis výpisu datumu dle 4-bytového pole (pack format)
**time** <regB> | popis výpisu času dle 3-bytového pole (unpack format)
**time** <regL> popis výpisu datumu dle 4-bytového pole (pack format)

**<base>** ::= **Dec** | **Hex** | **Bin** | **Chr** | dummy
*baze výpisu (dekad/hexadec/binar/znak/datum/čas/dekad)*
**<chlen>** ::= <const_expression> *celkový počet znaků (doplnění mezerami vlevo)*
**<frlen>** ::= <const_expression> *celkový počet desetinných míst reálného čísla*

**<edit_description>** ::= **edit** <base><regVBWIL>:<chlen>,<edit_lowlimit>,<edit_highlimit> |
**edit** <base><regR>:<chlen>:<frlen>,<edit_lowlimit>,<edit_highlimit> |
**edit** <regS>:<chlen>
**<editp_description>** ::= **editp** <regS>:<chlen>
<**edit_lowlimit**> ::= <const_expression>
<**edit_highlimit**> ::= <const_expression>

**<editenter_description>** ::= **editenter** <editenter_statement>
**<editenter_statement>** ::= <statement> *příkazy provedené po úspěšném ukončení editace*

**<point_description>** ::= **point** <x1>,<y1>
**<line_description>** ::= **line** <x1>,<y1>,<x2>,<y2> | *kreslení přímky zadané dvěma body*
**line rel**, <x1>,<y1>,<dx>,<dy>
*kreslení přímky zadané poč.bodem a relat.souř.konc.bodu*
**<x1>** ::= <RegnumC_Index>
**<y1>** ::= <RegnumC_Index>
**<x2>** ::= <regnumC_Index> *koncová souřadnice x*
**<y2>** ::= <regnumC_Index> *koncová souřadnice y*
**<dx>** ::= <regnumC_Index> *relativní koncová souřadnice x*
**<dy>** ::= <regnumC_Index> *relativní koncová souřadnice y*
**<w>** ::= <regnumC_Index> *šířka (x2-x1)*
**<h>** ::= <regnumC_Index> *výška (y2-y1)*

**<circle_description>** ::= **circle** <x1>,<y1>,<r> *kreslení kruhu, zadaného středem a poloměrem*
**<r>** ::= <regnumC_Index> *poloměr kruhu*
**<rect_description>** ::= **rect** <x1>,<y1>,<x2>,<y2> | *kreslení obdélníku zadaného horním levým a pravým spodním bodem absolutně*
**rect rel**, <x1>,<y1>,<dx>,<dy>
*kreslení obdélníku zadaného horním levým bodem a relat.souř.pravého dolního bodu*

**<fill_description>** ::= **fill** <x1>,<y1>,<x2>,<y2> | *kreslení vyplněného obdélníku zadaného horním levým a pravým spodním bodem absolutně*
**fill rel**, <x1>,<y1>,<dx>,<dy> *kreslení vyplněného obdélníku zadaného horním levým bodem a relat.souř.pravého dolního bodu*

**<graph_description>** ::= **graph** <graph_typedescription>,
**<**graph_number**>**,<graph_data>,<graph_min>,<graph max**>**,

```
                                <x1>,<y1>,<x2>,<y2>
                                kreslení grafu zadaného počtem bodů, prvním bytem dat, rozsahem hodnot
                                a souřadnicemi rohů obdélníka ve kterém bude vykreslen
<graph_typedescription>    ::=  <graph_type> | <graph_type>, rel
<graph_type>               ::=  | line | bar | point

<graph_number>             ::=  <RegnumC_Index>                                    počet bodů grafu
<graph_data>               ::=  <IndexRegister2>                          počáteční registr dat grafu
<graph_min>                ::=  <RegnumC_Index>                               minimální hodnota grafu
<graph_max>                ::=  <RegnumC_Index>                               maximální hodnota grafu

<graphxy_description>      ::=  graphxy <graphxy_typedescription>,
                                <graph_number>,<graphx_data>,<graphy_data>,<graphx_min>,<graphx_max>,<g
                                raphy min>,<graphy max>, <x1>,<y1>,<x2>,<y2>
                                kreslení grafu zadaného počtem bodů, prvním bytem dat x-ové i y-ové
                                složky, rozsahem hodnot a souřadnicemi rohů obdélníka ve kterém bude
                                vykreslen
<graphxy_typedescription>  ::=  <graph_type>  | <graph_type>, rel
<graphx_data>              ::=  <IndexRegister2>              počáteční registr dat x-ové složky grafu
<graphy_data>              ::=  <IndexRegister2>              počáteční registr dat y-ové složky grafu
<graphx_min>               ::=  <RegnumC_Index>                        minimální x-ová hodnota grafu
<graphx_max>               ::=  <RegnumC_Index>                        maximální x-ová hodnota grafu
<graphy_min>               ::=  <RegnumC_Index>                        minimální y-ová hodnota grafu
<graphy_max>               ::=  <RegnumC_Index>                        maximální y-ová hodnota grafu

<bar_description>          ::=  bar <bar_typedescription>, <bar_value>,<bar_lowvalue>,<bar_highvalue>,
                                <x1>,<y1>,<x2>,<y2> |          kreslení bargrafu do obdélníku zadaného
                                                        horním levým a pravým spodním bodem absolutně
                                bar <bar_direction>, rel, <bar_value>,<bar_lowvalue>,<bar_highvalue>,
                                <x1>,<y1>,<w>,<h>              kreslení bargrafu do obdélníku zadaného
                                                            horním levým bodem, šířkou a výškou

<bar_typedescription>      ::=  <bar_direction>                          upřesňující parametry bargrafu
<bar_direction>            ::=  top | bottom | left | right
                                                      strana obdélníka, od které se vykresluje bargraf
<bar_value>                ::=  <IndexRegister2>                                  zobrazovaná hodnota
<bar_lowvalue>             ::=  <RegnumC_Index>                 hodnota odpovídající počátku bargrafu
<bar_highvalue>            ::=  <RegnumC_Index>                   hodnota odpovídající konci bargrafu
```

### 1.9.3 Definice odezvy na stisk kláves na terminálu

```
<onkey_statements>         ::=  <onkey_statement> | <repeatonkey_statement>
                                                    provedení příkazů v <onkey_variant_statement>
<onkey_statement>          ::=  onkey <onkey_variants> end;                       reakce na stisk klávesy
<repeatonkey_statement>    ::=  repeatonkey <onkey_variants> end;
                                                          rychlejší reakce na opakovaný stisk kláves
<onkey_variants>           ::=  <onkey_variant>; { ; <onkey_variant } 

<onkey_variant>            ::=  <onkey_constants> : <onkey_variant_statement>

<onkey_constants>          ::=  <keyvalue> |
                                <keyvalue>,<onkey_constants>
<onkey_variant_statement>  ::=  <statement>

<wait_statement>           ::=  wait <timeout>, <nextscr_no>
                                 přechnod na obrazovku číslo <nextscr_no> po vyčerpání timeoutu (v sec)
<timeout>                  ::=  <const_expression>
<nextscr_no>               ::=  <const_expression>
```

```
Příklady:
configuration
  SWOBJ=TERM10, ADR=$300, NAME=T1, VAR=B10:5, SCRNO=1;        { po RESETu se zobrazí obrazovka č.1 }

terminal T1:1;                                                  { definice obrazovky č.1 }
begin
  bitmap 1; font 1;                  { výpis uživatelské podkladové bitmapy č.1, volba čísla fontu }
  position 2,2; print "ahoj";        { výpis textu od zadané posice naposledy zadaným číslem fontu }
  position 2,22; number B10:10; { výpis celočís.hodnoty, doplněné zleva mezerami na celkem 10 znaku }
```

```
  position 2,42; number R10:10:3;                    { výpis hodnoty typu real na 3 desetinná místa,
                                                       doplněné zleva mezerami na celkem 10 znaku }
  rect 1,1,200,100;              { nevyplněny obdélník horní levý roh 1,1, dolní pravý roh 200,100 }
  rect B1,B2,B3,B4;                { nevyplněny obdélník, jehož souřadnice jsou v registrech B1-B4 }
  bar bottom, R0, -1000, 1000, 200,1,210,100;     { bargraf, zobrazující odspodu hodnotu proměnné R0
                           s tim, ze spodní okraj odpovídá hodnotě -1000, horní okraj hodnotě 1000.
                                                       Plocha bargrafu není orámovaná obdélníkem ! }
  bar rect, left, R0, -1000, 1000, 200,1,210,100;
                                              { dtto, plocha bargrafu je navíc orámovaná obdélníkem }
  onkey
    '2',zF2:T1_ScrNo:=2;                          { po stisku klávesy "2" se zobrazí obrazovka č.2 }
    '1':B10:=B10+3;                     { po stisku klávesy "1" se hodnota registru B10 zvětší o 3 }
  end;
end;

terminal T1:2;                                                          { definice obrazovky č.2 }
begin
  bitmap 2; font 3;
  position 20,20; case B4 of
                  0..9: begin print "B4=",B4; circle 100,100,50; end;             { zobrazení textu,
                          hodnoty proměnné a kružnice v případě, že B4 má hodnotu v intervalu 0..9 }
                  else  begin print "B4<>0..9"; fill 100,100,120,120; end; { zobrazení textu, hodnoty
                         proměnné a plného obdélníku v případě, že B4 má hodnotu mimo interval 0..9 }
                  end;
  position 100,20;case B5 of                       { výpis textu v závislosti na hodnotě registru B5 }
                  0: print "t0";
                  1: print "t1";
                  2: print "t2";                    { když neplatí žádná varianta, nevypisuje se nic }
                  end;
  rect rel, 1,1,200,100;             { výpis obdélníku zadaného horním levým rohem, šířkou a výškou }
  onkey                                                  { reakce na stisknutí některých kláves }
    '1',zF1,zEsc:T1_ScrNo:=1;           { po stisku '1', ESC resp. F1 přechod zpět na obrazovku č.1 }
    '2':B10:=B10+2;                              { po stisku '2' se zvětší hodnota registru B10 o 2 }
    zF2:T1_Led:=$01;                          { po stisku F2 se rozsvítí nejlevější dioda na terminálu
                                              (nejnižší bit portu T1_Led odpovídá první diodě zleva) }
    zLe:T1_Led:=T1_Led shr 1;              { po stisknutí Left šipky se svítící dioda "posune" vlevo }
    zRi:T1_Led:=T1_Led shl 1;              { po stisknutí Left šipky se svítící dioda "posune" vpravo }
    end;
  wait 60, 1;                { přechod na obrazovku č.1 po 60 sec od posledního stisku lib. klávesy }
end;
```

# 1.10 Deklarace procedur:

## 1.10.1 Deklarace procedury

```
<procedure_declaration>   ::=  <procedure_heading> <procedure_body>
<procedure_heading>       ::=  procedure <userprocedure_name_def>
<userprocedure_name_def>  ::=  MAIN | INIT | ERROR | FAST10 |
                                                  definice uživatelské procedury se speciálním významem
                               <identifier>              místo definice jména jiné uživatelské procedury
<userprocedure_name>      ::=  <identifier>                    místo použití jména uživatelské procedury

<procedure_body>          ::=  <compound_statement>
<compound_statement>      ::=  begin <statements> end
<statements>              ::=  <statement>; { <statement> ; }

<statement>               ::=  <simple_statement> |
                               <structured_statement> |
                               <label_name_def> : <simple_statement> |
                               <label_name_def> : <structured_statement>
<label_name_def>          ::=  <identifier>                                    místo definice návěští
<label_name>              ::=  <identifier>                                     místo použití návěští
```

## 1.10.2 Jednoduché příkazy

```
<simple_statement>      ::=  <dummy_statement>                          prázdný příkaz (tj. "nic")
                             <assignment_statement> |                        přiřazovací příkaz
                             <procedure_statement> |                   příkaz volání procedury
                             <goto_statement> |              příkaz skoku na definované návěští
                             <exit_statement> |                     příkaz opuštění procedury

<dummy_statement>       ::=                                                        prázdný příkaz
<assignment_statement>  ::=  <IndexRegister> <assignment> <expression>          přiřazovací příkaz
<assignment>            ::=  := | =                                             znak pro přiřazení
```

*Hodnotu výsledku na základě hodnoty a typu <expression> definuje násl. tabulka* ***TAB2:***

| typ <register> | typ <expression> | rozsah <expression> | hodnota přiřazená do <register> |
|---|---|---|---|
| bit | bit | 0..1 | <expression> |
| bit | byte | 0 | 0 |
| bit | byte | >0 | 1 |
| bit | word | 0 | 0 |
| bit | word | >0 | 1 |
| bit | integer | 0 | 0 |
| bit | integer | <>0 | 1 |
| bit | longint | 0 | 0 |
| bit | longint | <>0 | 1 |
| bit | real | 0 | 0 |
| bit | real | <>0 | 1 |
| byte | byte | 0..255 | <expression> |
| byte | word | 0..255 | <expression> |
| byte | word | >255 | 255 1) |
| byte | integer | <0 | 0 1) |
| byte | integer | 0..255 | <expression> |
| byte | integer | >255 | 255 1) |
| byte | longint | <0 | 0 1) |
| byte | longint | 0..255 | <expression> |
| byte | 1longint | >255 | 255 1) |
| byte | real | <0 | 0 1) |
| byte | real | 0..255 | round(<expression>) |
| byte | real | >255 | 255 1) |
| word | byte | 0..255 | <expression> |
| word | word | 0..65535 | <expression> |
| word | integer | <0 | 0 1) |
| word | integer | 0..32767 | <expression> |
| word | longint | <0 | 0 1) |
| word | longint | 0..65535 | <expression> |
| word | longint | >65535 | 65535 1) |
| word | real | <0 | 0 1) |
| word | real | 0..65535 | round(<expression>) |
| word | real | >65535 | 65535 1) |
| integer | byte | 0..255 | <expression> |
| integer | word | 0..32767 | <expression> |
| integer | word | >32767 | 32767 1) |
| integer | integer | -32767..32767 | <expression> |
| integer | longint | <-32767 | -32767 1) |
| integer | longint | -32767..32767 | <expression> |
| integer | longint | >32767 | 32767 1) |
| integer | real | <-32767 | -32767 1) |
| integer | real | -32767..32767 | round(<expression>) |
| integer | real | >32767 | 32767 1) |
| longint | byte | 0..255 | <expression> |
| longint | word | 0..65535 | <expression> |
| longint | integer | -32767..32767 | <expression> |
| longint | longint | $-2^{31}+1..2^{31}-1$ | <expression> |
| longint | real | $<-2^{31}+1$ | $-2^{31}+1$ 1) |
| longint | real | $-2^{31}+1..2^{31}-1$ | round(<expression>) |
| longint | real | $>2^{31}-1$ | $2^{31}-1$ 1) |

| | | | |
|---|---|---|---|
| real | byte | 0..255 | <expression> |
| real | word | 0..65535 | <expression> |
| real | integer | -32767..32767 | <expression> |
| real | longint | $-2^{31}+1..2^{31}-1$ | <expression> |
| real | real | lib. | <expression> |
| string | string | lib. | <expression> |

*Pozn 1) Při přetečení při vyhodnocování se generuje RunTimeová chyba, zapisovaná do Error-Archivu (a případně následuje Reset celého systému).*

```
<procedure_statement>     ::=  <stdprocedure> |                          volání standardní procedury
                               <userprocedure_name>                     volání uživatelské procedury
<goto_statement>          ::=  goto <label_name>              skok na návěští, definované v proceduře
<exit_statement>          ::=  exit                        opuštění procedury (skok na konec procedury)
```

### 1.10.3 Strukturované příkazy

```
<structured_statement>    ::=  <compound_statement> | <conditional_statement> | <repetive_statement>
<conditional_statement>   ::=  <if_statement> | <case_statement>
<if_statement>            ::=  if <expression> then <statement> [else <statement>]
<case_statement>          ::=  case <expression> of <case_variants> [else <statement>] end

<case_variants>           ::=  <case_variant>; { <case_variant> ; }
                               <case_variant>                                              ::=
                                                          <case_constants> : <statement>

<repetive statement>      ::=  <repeat_statement> | <while_statement> | <for_statement>
<repeat statement>        ::=  repeat <statement> until <expression>
<while statement>         ::=  while <expression> do <statement>
<for statement>           ::=  for <Indexregister> <assigment> <expression> (to| downto) <expression>
                               [step <expression>]* do <statement> |
```

**** Není dosud implementováno***

### 1.10.4 Standardní procedury

*Všechny standardní procedury smějí mít jako operandy výhradně adresy registrů (včetně "malé" konstanty), překládají se jako speciální instrukce s předem daným počtem operandů.*

```
<stdprocedure>            ::=  <wait_procedure> |
                               <proc_procedure> |
                               <move_procedure> |
                               <moveSingle_procedure> |
                               <moveReal_procedure> |
                               <moveDouble_procedure> |
                               <inc_procedure> |
                               <dec_procedure> |
                               <string_procedures> |
                               <com_procedures> |
                               <archiv_procedures>

<wait_procedure>          ::=  WAIT                                  volání systémové procedury WAIT
<proc_procedure>          ::=  <proc_procedure_name> ( <IndexRegister>, <IndexRegister>,
                               <IndexRegister> )          volání systémové procedury PROC0 až PROC9
<proc_procedure_name>     ::=  PROC0 | PROC1 | PROC2 | PROC3 | PROC4 | PROC5 |
                               PROC6 | PROC7 | PROC8 | PROC9       systémové procedury PROC0 až PROC9

<move_procedure>          ::=  MOVE ( <register_source>, <register_destination>, <register_len> )
                               volání procedury move, s parametry - zdrojový registr, cílový registr a
                               počet přenesených byte
<moveSingle_procedure>    ::=  MOVESINGLE ( <register_source>, <register_destination> )  Procedura
                               převede hodnotu z realneho registru do 4 bytoveho realu, a naopak.
<moveReal_procedure>      ::=  MOVEREAL ( <register_source>, <register_destination> )
                               Procedura převede hodnotu z realneho registru do 6ti bytoveho realu, a
                               naopak.
<moveDouble_procedure>    ::=  MOVEDOUBLE ( <register_source>, <register_destination> ) Procedura
                               převede hodnotu z realneho registru do 8mi bytoveho realu, a naopak.

<register_source>         ::=  <IndexRegister> | <IndexRegister2>
```

**<register_destination>** ::= <IndexRegister> | <IndexRegister2>
*pokud je register typu int muze zde byt pouze IndexRegister2*

**<register_len>** ::= <expression>

**<inc_procedure>** ::= **INC (** <Register> **)**| **INC (** <Register>, <const_expression>**)** Procedura zvětší hodnotu registru o 1, popř. o hodnotu konst. výrazu.

**<dec_procedure>** ::= **DEC (** <Register> **)**| **DEC (** <Register>, <const_expression>**)** Procedura zmenší hodnotu registru o 1, popř. o hodnotu konst. výrazu.

*Následující procedury umožňují práci se stringovými proměnnými. Jsou zaměřeny zejména na kódování/dekódování zpráv.*

**<string_procedures>** ::= **AddS (** <regS_Destin> , <regS_Source> , <len> **)** |
*RegD:=RegD+RegS; {len je přitom délka RegD}*
**StrClr (** <regS> **)** | *nastavení aktuální délky stringu na 0*
**StrAddStr (** <regS_Destin> , <regS_Source> , <pos> , <val> **)** |
*RegSD:=copy(RegSD+copy(RegSS,pos,val),1,SizeOf(ResSD)-1)*
**StrAddVal (** <regS_Destin> , <regNumC> , <F1> **)** |
*RegSD:=copy(RegSD+stri(RegNumC,F1),1,SizeOf(ResSD)-1)*
**StrAddReal (** <regS_Destin> , <regR> , <F1> , <F2> **)** |
*RegSD:=copy(RegSD+strf(RegNumC,F1,F2),1,SizeOf(ResSD)-1)*
**StrPos (** <regS_Source> , <regS_Key> , <regB_pos> **)** |
*RegBpos:=pos(RegSKey,RegSSource)*
**StrVal (** <regS_Source> , <pos> , <len> , <RegNum>, <ErrorCode> **)** |
*val(copy(regSSource,pos,len),RegNum,ErrorCode)*

**<F1>** ::= <number> *počet znaků celkem při převodu <regNumC> na string*
**<F2>** ::= <number> *počet desetinných míst při převodu <regR> na string*
**<pos>** ::= <RegNumC> *pozice ve stringu, od které se dekóduje číslo*
**<len>** ::= <RegNumC> *počet znaků celkem při převodu části stringu na číslo*
**<ErrorCode>** ::= <RegNum> *registr, do kterého se zapisuje případný kód chyby*

*pozn.: Všechny funkce pro práci se stringy mimo AddS nejsou dosud implementovány.*

*Následující procedury umožňují úschovu a zpětné vyzvednutí dat z archivu.*

**<archiv_procedures>** ::= **ArchiveSave (** <ARCHIV_object_name> **)**
*uložení sady registrů do (dalšího) záznamu do archivu*
**ArchiveLoad (** <ARCHIV_object_name>,<Err_Flag> **)**
*načtení (dalšího) záznamu z archivu do sady registrů*
**ArchiveSaveNb (** <ARCHIV_object_name> **)**
*uložení sady registrů do (dalšího) záznamu do archivu*
**ArchiveLoadNb (** <ARCHIV_object_name>,<Err_Flag> **)**
*načtení (dalšího) záznamu z archivu do sady registrů*
**ArchiveLoadPrev (** <ARCHIV_object_name>,<Err_Flag> **)**
*načtení (dalšího) záznamu z archivu do sady registrů*
**ArchiveClearOld (** <ARCHIV_object_name> **)**
*vymazání nejstaršího jednoho záznamu archivu*
**ArchiveClearAll (** <ARCHIV_object_name> **)**
*vymazání archivu, příští zápis do 1. záznamu*
**ArchiveSeekFirst (** <ARCHIV_object_name> **)**
*nastavení čtení z archivu z 1. záznamu*
**ArchiveSeekLast (** <ARCHIV_object_name> **)**
*nastavení čtení z archivu z posledního záznamu*
**ArchiveSeek ( <ARCHIV_object_name>, <Register> )**
*nastavení čtení z archivu na položku s nejbližším nižším nebo stejným datumem jako v zadaném registru, zatím nepimplementováno*

*Následující procedury umožňují úschovu a zpětné vyzvednutí dat z archivu.*

**<archiv_functions>** ::= **ArchiveCount (** <ARCHIV_object_name> **)**
*funkce - poskytnutí počtu platných záznamů v archivu*

*Následující procedury a funkce umožňují práci se SWObjektem tabulka - nastavovávání hodnot, třídění a vybírání podle prvního vektoru třemi různými metodami.*

**<TAB_procedures>** ::= **SetTab (** <TAB_object_name> , <settab_parameters> **)**
*nastavení hodnot tabulky, zadává se vždy několik dvojic prvků x a y, podle jejich počtu se nastaví hodnota TAB_NB; po naplnění se tabulka automaticky setřídí.*
**SortTab (**<TAB_object_name> **)** *setřídění tabulky*
**AddTab (**<TAB_object_name>, <settab_parameter> **)** *setřídění tabulky*

**DelTab (**<TAB_object_name> **)** *setřídění tabulky*

**<settab_parameters>** ::= <settab_parameter> { , <settab_parameter> ) *dvojice x a y prvků tabulky*

**<settab_parameter>** ::= **[** <expression> , <expression> **]** dvojice x a y prvků tabulky

**<TAB_functions>** ::= **GetTab (** <TAB_object_name> , <gettab_parameter>, <expression> **)**
*procedura pro výběr prvků z tabulky třemi přibližnými metodami*

**GetEkvTab (** <TAB_object_name>, <expression> **)**
*procedura pro výběr prvku z tabulky, který je přepočítaný vzhledem k parametru plnění*

**<gettab_parameters>** ::= **LOM** | **POM** | **LM**
*parametr výběru prvku z tabulky - Levá a Pravá Obdélníková metoda, Lichoběžníková metoda*

*Procedury pro ovládání sériové komunikace COM*

**<COM_procedures>** ::= **ComOn (** <COM_name> [ , <IndexRegister2> ]**)** *procedura aktivuje COM*
**ComOff (** <COM_name> **)** *deaktivace COMu*
**ComSend (** <COM_name> ,<IndexRegister> **)** *vyslání zprávy v registru IndexRegister po komunikačním kanálu*
**ComReceive (** <COM_name> ,<IndexRegister> **)** *přijetí zprávy po komunikačním kanáludo registru IndexRegister*
**ComRecFlush (**<COM_name> **)** *Smazání bufferu kom. kanálu Následující procedury umožňují pracovat s registry typu word jako s timery;*
*Tzn. po aktivaci se budou každých 10ms/1s {podle typu} zvyšovat o 1, příkaz timeroff přičítání ukončí.*
**<TIMER_procedures>** ::= **TimerOn (** <RegW> , <TIMER_types> **)**
*procedura aktivuje RegW jako timer daného typu*
**TimerOff (** <RegW> **)**
*deaktivace timeru*

**<TIMER_types>** ::= **per10ms** | *typ pro deseti milisekundový timer*
**per1s** *typ pro sekundový timer*

## 1.10.5 Výrazy

**<expression>** ::= <simple_expression> { <relational operators> <simple_expression> }
*výraz*

**<simple expression>** ::= <term> { <adding_operators> <term> }
**<term>** ::= <faktor> { <multiplying_operators> <faktor> }
**<faktor>** ::= <Indexregister> | *jméno registru (i symbolické)*
<const_factor> | *konstanta (číselná, znaková, i symbolická)*
**(** <expression> **)** | *libovolný výraz uzavřený do závorek*
<stdfunction> | *volání standardní funkce*
<signum><faktor> | *unární "+" resp. "-" operace*
**not** <faktor> *unární "not" operace*

*<expression> se vyhodnocuje při provádění programu. Výsledná hodnota a typ je dána dle pravidel, uvedených v **TAB3:***

| způsob zadání <const_expression> resp. <expression> | rozsah 1.operandu | rozsah 2.operandu | typ výsledku |
|---|---|---|---|
| <not><faktor> | bit | | bit |
| | byte | | byte |
| (unární operátor **not**) | word | | word |
| | jiný typ | | **undef** |
| <signum><faktor> | byte | | integer |
| | word | | longint |
| (unární operátor + a -) | integer | | integer |
| | longint | | longint |
| | real | | real |
| <faktor><mult_math_operators><faktor> | byte | byte | byte |
| <term><add_math_operators><term> | byte | word | word |
| | byte | integer | integer |
| (operátory ***,/,+,-** ) | byte | longint | longint |
| | byte | real | real |

| | | | |
|---|---|---|---|
| | word | byte | word |
| | word | word | word |
| | word | integer | integer |
| | word | longint | longint |
| | word | real | real |
| | integer | byte | integer |
| | integer | word | integer |
| | integer | integer | integer |
| | integer | longint | longint |
| | integer | real | real |
| | longint | byte | longint |
| | longin | word | longint |
| | longint | integer | longint |
| | longint | longint | longint |
| | longint | real | real |
| | real | lib.čísel.typ | real |
| | ostatní kombinace | | **undef** |
| <term><add_log_operators><term><br><br><faktor><mult_log_operators><faktor><br><br>(operátory **xor,or,and**) | bit | bit | bit |
| | byte | byte | byte |
| | byte | word | word |
| | word | byte | word |
| | word | word | word |
| | ostatní kombinace | | **undef** |
| <faktor><mult_shift_operators><faktor><br><br>(operátory **shl** a **shr**) | byte | byte | byte |
| | word | byte | word |
| | ostatní kombinace | | **undef** |
| <faktor><mult_div_operators><faktor><br><br>(operátor **mod**) | byte | byte | byte |
| | byte | word | word |
| | byte | integer | integer |
| | byte | longint | longint |
| | word | byte | word |
| | word | word | word |
| | word | integer | integer |
| | word | longint | longint |
| | integer | byte | integer |
| | integer | word | integer |
| | integer | integer | integer |
| | integer | longint | longint |
| | longint | lib.čísel.typ | longint |
| | ostatní kombinace | | **undef** |
| <simple><relational_operators><simple><br><br>(<simple> <=\|<\|<>\|=\|>=\|> <simple>) | lib.čísel.typ | lib.čísel.typ | bit |

### 1.10.6 Standardní funkce

*Všechny standardní funkce smějí mít jako číselné operandy adresy registrů (včetně "malé" konstanty), nebo výrazy, překládají se jako speciální instrukce s předem daným počtem operandů.*

**<stdfunction>** ::= <system_function>
<math_function>
<realmath_function>
<tab_function>

**<system_function>** ::= **SIZEOF (** <regCmn> **)** *velikost registru ze spol. adr.prostoru v bytech*
**LENGTH (** <regS> **)** *aktuální délka stringu v registru typu RegS (String)*
**PACKTIME (** <register> **)** zapakuje čas zadaný v registru
**UNPACKTIME (** <register> **)** rozpakuje čas zadaný v registru
**CRC8 (**<regVBWILD> , <regVBWILD> **)** výpočet crc8 přes blok adres
**CRC16 (**<regVBWILD> , <regVBWILD> **)** výpočet crc16 přes blok adres
**ADDR (**<register>**)** adresa daného registru

**<math_function>** ::= **ABS (** <expression> **)** *absolutní hodnota*
**SGN (** <expression> **)** *znaménko hodnoty (-1,0,1)*
**MIN (** <expression> , <expression> **)** *minimum ze dvou hodnot*
**MAX (** <expression> , <expression> **)** *maximum ze dvou hodnot*

**<realmath_function>** ::= **SIN (** <expression> **)** sinus výrazu
**COS (** <expression> **)** cosinus výrazu

```
                          LN  ( <expression> )                  přirozený logaritmus výrazu
                          LOG ( <expression> )                  dekadický logaritmus výrazu
                          EXP ( <expression> )                    přirozený exponent výrazu
                          SQR ( <expression> )                         druhá mocnina výrazu
                          SQRT( <expression> )                             odmocnina výrazu

<tab_function>      ::=   GetTab ( <Tabulka>, <gettab_parameters>, <RegNumC> )
                          GetEkvTab ( <Tabulka>, <RegNumC>, <RegNumC> )
```

*V následující tabulce **TAB4** jsou pro jednotlivé standardní funkce uvedeny přípustné typy argumentů a typ výsledné funkční hodnoty:*

| název funkce | typ 1. parametru | typ 2. parametru | typ 3. parametru | typ výsledku |
|---|---|---|---|---|
| SIZEOF | libovol.typ | - | - | byte |
| LENGTH | string | - | - | byte |
| ABS | integer | - | - | word |
| ABS | longint | - | - | longint |
| SGN | lib.čísel.typ | - | - | integer |
| MIN, MAX | lib.čísel.typ | lib.čísel.typ | - | nadrozsah obou typů |
| SIN, COS, LN, LOG, EXP, SQR, SQRT, RND | lib.čísel.typ | - | - | real |
| GETTAB | tabulka | parametr aproximace | registr | registr |
| GETEKVTAB | tabulka | registr | registr | registr |
| ARCHIVSIZE | archiv | - | - | byte |
| CRC8 | registr | registr | - | byte |
| CRC16 | registr | registr | - | word |

# 1.11 Seznam klíčových slov a standardních identifikátorů:

## 1.11.1 Globální klíčová slova

*#define, #include, and, begin, bit, byte, case, configuration, constant, datetime, do, downto, else, end, exit, for, goto, if, integer, mainexit, of, options, or, procedure, real, repeat, string, terminal, then, to, shl, shr, step, symbol, xor, until, while, word*

## 1.11.3 Klíčová slova v deklaraci TERMINAL

*bar, bin, bitmap, circle, date, dec, edit, fill, font, help, hex, line, onkey, point, position, print, rect, rel, time, wait,*
*POM, LOM, LM,*
*bottom, left, right, top,*
*MS, S*
*zCr, zBs, zSBs, zEsc, zSpace zDel, zUp, zDn, zLezRi, zF1, zF2, zF3, zF4, zF5, F6, zF7, zF8, zF9, zF10, zHome, zPgUp, zEnd, zPgDn, zIns, zTab*

## 1.11.4 Klíčová slova v deklaraci CONFIGURATION

*a, addr, adr, auto, archiverror, archivfirst, archivlast, b, beep, bit8, bit10, bit12, bit14, bit16, buff, clock, c, com, d, db, dgap, du, egap, fast10, filter1, filter2, genstr, hwobj, igap, ioadda01, ioaterm10,iodio01, iodoo01, iodterm10,iodxo01, ioterm10, kp, ki, kd, led, main, maxu, maxy, minu, miny, name, ni1000, iopbus, par, per10ms, per1s, pid, pidr, pt100, pt500, recbuff, save, saver, scrno, start, stop, sendbuff, swobj, tab, term01, term03, term10, term10a, termp, terms, timer10, ts, in0, in1, in2, in3, in4, in5, in6, in7, in8, in9, in10, in11, in12, in13, in14, in15, ina, inb, inc, ind, int, out0, out1, out2, out3, out4, out5, outa, outb, outc, outd, relk, sb, su, u, uman, var, w, x, y*

## 1.11.5 Standardní funkce a procedury, použitelné v PROCEDURE a TERMINAL

*abs, cos, crc8, crc16, exp, length, ln, log, max, min, rnd, sin, sizeof, sgn, sqr, sqrt, ArchiveSize, GetEkvTab, GetTab, PackTime, UnPackTime*

*AchiveClearAll, ArchiveClearOld, ArchiveCount, ArchiveLoad, ArchiveLoadPrev, ArchiveSeekFirst, ArchiveSeekLast, ArchiveSave, ArchiveLoadNb, ArchiveSaveNb, ComOn, ComSend, ComReceive, ComRecFlush, ComOff SetTime,PackTime,UnpackTime, PulseOn, PulseOff, TimerOn, TimerOff, SortTab, SetTab, Move, MoveSingle, MoveReal, MoveDouble*

### 1.11.6 Uživatelské procedury se speciálním významem

**INIT** prováděna po RESETu systému
**FAST** prováděna cyklicky každých 10 ms pod přerušením
**MAIN** prováděna cyklicky na pozadí. V této proceduře může být použita speciální standardní procedura **WAIT** a **PROCx**.

# 2. Implementace jazyka KIT-BASIC

## 2.1 Základní parametry překladače a Run-time systému

| **verze** | **Compiler MS - DOS** | **Compiler Windows** | **Procesor** |
|---|---|---|---|
| Maximální délka zdrojového programu v bytech | neomezeno | neomezeno | - |
| Maximální počet řádků zdroj. programu | $7FFFFFF | $7FFFFFF | - |
| Maximální délka 1 řádky zdrojového programu | $7FFFFFF | $7FFFFFF | - |
| Maximální výsledná délka display sekce v bytech | 52000 | 65520 | - |
| Maximální výsledná délka conf sekce v bytech | 5000 | 5000 | - |
| Maximální výsledná délka P-kódu v bytech | 58000 | 65520 | - |
| Maximální počet použitých identifikátorů | 2000 | 2500 | - |
| | | | - |
| Velikost prostoru celočíselných registrů | 2000 bytů | 2000 bytů | - |
| Velikost prostoru registrů typu real | 250 bytů | 250 bytů | - |
| | | | |
| Počet instrukcí p-codu, vygenerovaných na jeden příkaz jazyka | cca | cca | - |
| Poměr délky p-codu ku délce zdrojového programu bez komentářů | cca | cca | - |
| | | | |
| Maximální počet instrukcí p-codu v proceduře MAIN | neomezeno | neomezeno | - |
| Počet provedených instrukcí p-codu za 1 sec (KITV40 8MHz) | - | - | cca |
| Počet provedených instrukcí p-codu za 1 sec (KITV40 16MHz) | - | - | cca |
| Maximální počet instrukcí p-codu v proceduře FAST10 | neomezeno | neomezeno | - |
| Maximální počet instrukcí p-codu v proceduře INIT | neomezen | neomezen | - |
| | | | - |
| Maximální počet HW-objektů PBUS | neomezeno | neomezeno | - |
| Maximální počet HW-objektů IODIO01 | neomezeno | neomezeno | - |
| Maximální počet HW-objektů IODOO01 | neomezeno | neomezeno | - |
| Maximální počet HW-objektů IODXO01 | neomezeno | neomezeno | - |
| Maximální počet HW-objektů IOTERM10 | neomezeno | neomezeno | - |
| Maximální počet HW-objektů ADDA | neomezeno | neomezeno | - |
| Maximální počet HW-objektů COM | 2 | 2 | - |
| Maximální počet HW-objektů TERM01 | 0 | 0 | - |
| Maximální počet HW-objektů TERM03 | 0 | 0 | - |
| Maximální počet HW-objektů TERM10 / TERM10A | 1 | 1 | - |
| | | | |
| Max. počet zapnutých timerů | - | - | 256 |
| Maximální počet SW-objektů TAB | 10 | 10 | - |
| Maximální počet SW-objektů FILTER | 0 | 0 | - |
| Maximální počet SW-objektů PIDR | 10 | 10 | - |
| Maximální počet SW-objektů PID | 10 | 10 | - |
| Maximální počet SW-objektů SAVER | neomezeno | neomezeno | - |
| Maximální celkový počet intervalů v SAVER objektech | 50 | 50 | - |

| | | | |
|---|---|---|---|
| Maximální počet SW-objektů ARCHIV | 10 | 10 | - |
| Maximální počet intervalů v ARCHIVU | 10 | 10 | - |
| Maximální počet SW-objektů GENSTR | 3 | 3 | - |
| | | | |
| Maximální počet analogových desek | 5 | 5 | - |
| Maximální počet desek IOTERM10 | 1 | 1 | - |
| | | | |
| Maximální počet binárních vstupů / výstupů | 40 | 40 | - |
| Maximální počet analogových vstupů | 40 | 40 | - |
| Maximální počet analogových výstupů | 10 | 10 | - |
| Maximální počet prvků v sekci case u IOTERM10 | 27 | 27 | - |
| | | | |
| Maximální počet obrazovek jednoho terminálu | 255 | 255 | - |
| | | | |
| Velikost zásobníku pro výrazy P-kódu | - | - | 6x99 bytů |
| Max počet polozek v Error archivu | 50 | 50 | - |

## 2.2 Časová osa provádění jednotlivých procesů

Rozložení provádění procesů v čase:
1) Pod přerušením každých 10 (popř. 20, 50) ms se provádí systémová obsluha **SYSTEM10**, skládající se z
- systémové rutiny pro obsluhu registrů TIMER 10ms,
- systémové rutiny pro obsluhu vstupů, definovaných jako **FAST**
- p-kod uživatelské procedury **FAST**
- systémové rutiny pro obsluhu výstupů, definovaných jako **FAST**

2) Na pozadí se provádí algoritmus těchto procesů:
- **SYSTEMBGRND**
- **TERMINAL** (pro každý SWOBJ jeden proces)
- **PID** regulátory (pro každý SWOBJ jeden proces)

3) Proces **SYSTEMBGRND** se skládá z těchto kroků:
- systémové rutiny pro obsluhu SWOBJ= (v této verzi žádný)
- systémové rutiny pro obsluhu vstupů, definovaných jako **MAIN**
- p-kod uživatelské procedury **MAIN**
- p-kod z právě aktivní obrazovky každého terminálu při stisku klávesy resp. vyčerpaní timeoutu dle popisů **onkey** resp. **wait**
- systémové rutiny pro obsluhu výstupů, definovaných jako **MAIN**
- další systémové služby, např. občerstvení watchdogu, aby se neprovedl RESET ap.

Procedura **SYSTEMBGRND** je pravidelně přerušována prováděním procedur **SYSTEM10** resp. **SYSTEM100.**