

**Řídicí systémy TRONIC 2000**

**TRONIC CONTROL s. r. o.**

# **Příručka programátora v jazyce LEDA**

Ing. Dušan Kliment

Ing. Pavel Lašťovka

Revize 1.3 – duben 2001

únor 1994

prosinec 1999

Obsah:

<b>1. ÚVOD</b>	<b>5</b>
<b>2. CHARAKTERISTIKA JAZYKA</b>	<b>6</b>
<b>3. POPIS JAZYKA LEDA</b>	<b>8</b>
<b>3.1 KOMENTÁŘE</b>	<b>8</b>
<b>3.2 PROMĚNNÉ</b>	<b>8</b>
3.2.1 TYPY PROMĚNNÝCH	8
3.2.2 IMPLICITNÍ PROMĚNNÉ	9
3.2.2.1 Vstupní a výstupní pole	9
3.2.2.2 Specifikace polí implicitních proměnných	10
3.2.3 SYSTÉMOVÉ PROMĚNNÉ	10
3.2.4 PROMĚNNÉ DEKLAROVANÉ V UŽIVATELSKÉM PROGRAMU	11
3.2.4.1 Deklarace proměnných	11
3.2.4.2 Deklarovaná pole	12
3.2.4.3 Ekvivalence alokace polí	12
3.2.5 IDENTIFIKÁTORY	12
3.2.6 KONSTANTY	12
3.2.7 REPREZENTACE	13
3.2.7.1 Standardní reprezentace	15
3.2.8 OPERÁTORY	17
3.2.9 VÝRAZY	17
3.2.10 PŘÍKAZY	19
3.2.10.1 Přiřazovací příkaz	19
3.2.10.2 Podmíněný příkaz	20
3.2.10.3 Příkazy cyklu	20
3.2.10.4 Příkaz SWI	21
3.2.10.5 Příkaz BREAK	22
3.2.10.6 Příkaz CONT	22
3.2.10.7 Příkaz RET	22
3.2.10.8 Příkaz GOTO	22
3.2.11 SLOŽENÝ PŘÍKAZ	22
3.2.12 NÁVĚŠTÍ	22
3.2.13 STANDARDNÍ FUNKCE	22
3.2.14 UŽIVATELSKÉ FUNKCE	26
3.2.15 BLOKY	28
3.2.16 SÍTĚ	28
3.2.17 EKVIVALENCE ŘETĚZCŮ	28
3.2.18 PROGRAM	29
3.2.19 DIREKTIVA INCLUDE	29
<b>3.3 PRÁCE SE ZNAKOVÝMI PROMĚNNÝMI, POLI A ŘETĚZCI</b>	<b>30</b>
<b>3.4 PODOBNOST A ROZDÍLY LEDA A JAZYKA C</b>	<b>32</b>
<b>4. POPIS STANDARDNÍCH FUNKCÍ</b>	<b>35</b>

4.1 MAX, MIN - VÝBĚR OKAMŽITÉHO MAXIMA, RESP. MINIMA	35
4.2 CMP - KOMPARÁTOR	35
4.3 SEL - ŘÍZENÝ VÝBĚR ALTERNATIVNÍCH SIGNÁLŮ	36
4.4 FIL - FILTR 1. ŘÁDU	37
4.5 SUM - SUMACE ČASOVÉ ŘADY	38
4.6 PID REGULÁTOR	38
4.7 PDC REGULÁTOR	41
4.8 SET, RES - NAHOZENÍ, RESP. SHOZENÍ, DVOUHODNOTOVÉ PROMĚNNÉ	43
4.9 DSR, SDR - ZPOŽDĚNÉ NAHOZENÍ, RESP. SHOZENÍ, BOOL VÝSTUPU	44
4.10 IMP, NIMP - IMPULS OD NÁBĚŽNÉ HRANY, IMPULS OD SESTUPNÉ HRANY	45
4.11 DIF - 1. DIFERENCE RESP. INDIKACE ZMĚNY	45
4.12 MSF - MONOSTABILNÍ KLOPNÝ OBVOD	46
4.13 NLT - NELINEÁRNÍ TRANSFORMACE	47
4.14 CNT - ČÍTAČ	48
4.15 DBN - PÁSMO NECITLIVOSTI	49
4.16 SAT - SATURACE	49
4.17 MAV - KLOUZAVÝ PRŮMĚR	50
4.18 MMA RESP. MMI - MAXIMUM RESP. MINIMUM NA KLOUZAVÉM INTERVALU	50
4.19 TDL - DOPRAVNÍ ZPOŽDĚNÍ	51
4.20 SQT - ODMOCNINA	52
4.21 ABS - ABSOLUTNÍ HODNOTA	52
4.22 IPR - VSTUPNÍ NORMALIZACE ANALOGOVÝCH SIGNÁLŮ	52
4.23 OPR - VÝSTUPNÍ NORMALIZACE ANALOGOVÝCH SIGNÁLŮ	53
4.24 PIM - PULZNÍ INTERVALOVÝ MODULÁTOR	54
4.25 REC - ZÁPIS DO CYKlickÉHO ZÁSOBNÍKU	56
4.26 CER - ČTENÍ DAT Z CYKlickÉHO ZÁSOBNÍKU	57
4.27 OUTF - VÝSTUP DO ZNAKOVÝCH SOUBORŮ A NA TISKÁRNU	57
4.28 OUTS - GENERACE ZNAKOVÉHO POLE ČÍSELNÝCH ÚDAJŮ	58
4.29 OUTF - INICIALIZACE VYSLÁNÍ ZNAKOVÉ ZPRÁVY	59
4.30 ATN - PŘEVOD POLE ZNAKŮ NA ČÍSELNOU HODNOTU	59
4.31 SRG - POSUVNÝ REGISTR	59
4.32 CPS - SDRUŽOVÁNÍ SKUPIN BOOL PROMĚNNÝCH	60
4.33 DCPS - ROZKLAD PROMĚNNÝCH NA SKUPINY BOOL	60
4.34 BEEP - TÓNOVÝ VÝSTUP	61
4.35 AUDIO - OBECNÝ TÓNOVÝ VÝSTUP	61
4.36 WRITEC - ZÁPIS ZÁZNAMU DO ARCHIVNÍHO DISKOVÉHO SOUBORU	62
4.37 READC - ČTENÍ ZÁZNAMŮ DANÉHO ČAS.INTERVALU Z ARCHIV.SOUBORU	63
4.38 AKTC - AKTUALIZACE BINÁRNÍHO ARCHIVNÍHO SOUBORU	64
4.39 LASTTC - NAČTENÍ DATA A ČASU POSLEDNÍHO ZÁZNAMU ARCHIV.SOUB.	65
4.40 PUTARR, GETARR	66
4.40.1 PŘENOS DAT PO SBĚRNICI AMICAN	67
4.40.2 PAMĚŤ SE SEKVENČNÍM PŘÍSTUPEM PRO ZÁZNAMY DAT	68
4.40.2 UNIVERZÁLNÍ ZOBRAZOVACÍ A ZADÁVACÍ BOD DISPLEJE	69
4.40.3 NASTAVENÍ SYSTÉMOVÉHO ČASU Z UŽIVATELSKÉHO PROGRAMU	70
4.40.4 ULOŽENÍ KONSTANT DO PAMĚTI FLASH ROM	71
4.40.5 OVLÁDÁNÍ PARALELNÍ TISKÁRNY (POUZE T2008S)	71
4.40.6 ULOŽENÍ A VYTAŽENÍ DAT Z BUFFERU PAMĚTI	71
4.40.6 KOMUNIKACE PO TELEFONNÍCH LINKÁCH POMOCÍ MODEMŮ	72
4.40.7 KOMUNIKACE PO SBĚRNICI MODBUS RTU	74
<b>5. POPIS STANDARDNÍCH REPREZENTACÍ</b>	<b>75</b>

5.1 STANDARDNÍ REPREZENTACE BEZROZMĚRNÝCH VELIČIN	75
5.2 STANDARDNÍ REPREZENTACE FYZIKÁLNÍCH VELIČIN	76
5.3 REPREZENTACE PRO FORMÁTOVÁNÍ ČÍSELNÝCH ÚDAJŮ	77
5.4 \$D - ZOBRAZENÍ HODNOT V DIGITÁLNÍ FORMĚ	78
5.5 \$CD - ZOBRAZENÍ HODNOT V DIGITÁLNÍ FORMĚ S ŘÍZENOU BARVOU	79
5.6 \$BAR - SLOUPCOVÝ GRAF	80
5.7 \$TIGR - GRAF ČASOVÝCH PRŮBĚHŮ ANALOGOVÝCH SIGNÁLŮ	81
5.8 \$TICUR - GRAF PRŮBĚHŮ ANALOG. SIGNÁLŮ S ÚDAJI ABSOLUTNÍHO ČASU	84
5.9 \$XYCUR - ZOBRAZENÍ GRAFŮ V SOUŘADNICÍCH X,Y	86
5.10 \$TAB - ZOBRAZENÍ TABULKY HODNOT	87
5.11 \$TEXT - ZOBRAZENÍ ALTERNATIVNÍCH TEXTOVÝCH ŘETĚZCŮ	88
5.12 \$MTEXT - REPREZENTACE SKUPINY PROMĚNNÝCH TEXTOVÝMI ŘETĚZCI	89
5.13 \$GTEXT - ZOBRAZENÍ ŘETĚZCE JAKO PRVKU POLE ŘETĚZCŮ	89
5.14 \$MAC - ZOBRAZENÍ ALTERNATIVNÍCH MAKROSYMBOLŮ	90
5.15 \$MMAC - REPREZENTACE SKUPINY PROMĚNNÝCH GRAF. MAKROSYMBOLY	90
5.16 \$GMAC - ZOBRAZENÍ GRAF. MAKROSYMBOLU URČENÉHO INDEXEM	91
5.17 \$READ - ZOBRAZENÍ GRAFICKÉHO OBRAZU	91
<b>6. VOLÁNÍ STANDARDNÍCH FUNKCÍ A REPREZENTACÍ - PŘEHLED FORMÁTŮ</b>	<b>92</b>
<b>7. PŘÍKLADY REALIZACE FUNKČNÍCH CELKŮ</b>	<b>99</b>
<b>8. SOUHRN UPOZORNĚNÍ, RAD A DOPORUČENÍ</b>	<b>104</b>
8.1 ŘAZENÍ SÍTÍ A VIDITELNOST PROMĚNNÝCH	104
8.2 VOLBA PERIOD ZPRACOVÁNÍ DAT	105
8.3 RYCHLOST INTERPRETACE	106
8.4 INICIALIZACE PROMĚNNÝCH	106
8.5 VÝMĚNA DAT MEZI STANICÍ MASTER A SLAVE	107
8.6 LADĚNÍ APLIKAČNÍCH PROGRAMŮ	108

# 1. Úvod

Cílem tohoto materiálu je podat stručný popis programovacího jazyka LEDA, který byl ve své prvotní verzi definován autorem tohoto materiálu v průběhu roku 1989 v návaznosti na úkoly vývoje modulárních řídicích systémů DERIS 900 a TRONIC 2016 řešené v oddělení vývoje základního programového vybavení systémů pro automatické řízení technologických procesů ve Výzkumném ústavu automatizačních prostředků.

Jazyk LEDA se v mnohém podobá jiným specializovaným jazykům podobného zaměření, které používají při přípravě aplikačních programů pro své řídicí systémy různé zahraniční firmy, i jazykům pro univerzální použití; větší podobnost lze nalézt např. mezi LEDA a jazykem SPL firmy Johnson Controls.

Některé specifické vlastnosti jazyka LEDA vycházejí z tradičních rysů koncepce uživatelského programování systémů DASOR 600 a DASOR 601 vyvinutých v minulosti na stejném pracovišti ve VÚAP.

Vývoj jazyka nelze považovat za ukončený; předpokládá se, že další verze budou vznikat na základě zkušeností získaných při jednotlivých aplikacích řídicích systémů TRONIC řady 2008 a 2032.

Následující popis jazyka LEDA není učebním textem, nenahrazuje také popis implementace na konkrétní přístrojové vybavení a nezahrnuje popis užití programových komplexů COLEDA a WINLEDA, které vytvářejí na osobních počítačích uživatelské prostředí pro vývoj aplikačních programů psaných v jazyce LEDA.

Autorem několika variant překladače jazyka LEDA je RNDr. Julius Janovský; autorem preprocesoru těchto překladačů je RNDr. František Mráz (viz Úvod). Autorem komplexu COLEDA je Ing. Tomáš Teska; autorem tzv. "interpreteru reprezentací" je Ing. Aleš Drbohlav. Autorem komplexu WINLEDA je Ing. Pavel Lašťovka. Autorem programových driverů pro různé typy procesorových jednotek systémů TRONIC je Ing. Pavel Lašťovka a autorem interpreterů exekutiv pro tyto jednotky a pro PC je Ing. Dušan Kliment.

Vývojem tohoto programového vybavení se v minulých letech zabývala společnost Auxilia Automatica s.r.o., která zřídila právo distribuce pro s.p. INTHERM Praha (později INTHERM Praha s.r.o.), a.s. Metra Blansko, a.s. Regula Praha a TRONIC CONTROL s.r.o.. Od začátku roku 1993 je další vývoj a distribuce programového vybavení předmětem činnosti firmy Automatica Amiga s.r.o., k jejímž dalším odběratelům patří mj. ZVVZ Milevsko a.s., PRO 8 Hradec Králové a Unicontrols s.r.o. Praha (implementace na systémy firmy PEP Modular Computers).

Připomínky k tomuto materiálu, další dokumentaci navazujícího programového vybavení i k programovému vybavení samotnému, lze zasílat na adresu:

**Ing. Pavel Lašťovka – vedoucí vývoje**

**Tronic Control s.r.o.**

**Dělnická 27**

**170 00 Praha 7**

**tel./fax: 266 710 254-5, 266 772 218, 266 772 214**

**[paha@tronic.cz](mailto:paha@tronic.cz), [www.tronic.cz](http://www.tronic.cz)**

## 2. Charakteristika jazyka

Jazyk LEDA je jazykem vyšší úrovně - strojově nezávislým - určeným speciálně pro řešení úloh zpracování dat získávaných z technologického procesu, pro generaci akčních zásahů na základě algoritmů zpětnovazebního spojitého a dvouhodnotového řízení (regulace a sekvenční automaty) a pro zpracování povelů operátora technologického procesu.

Programy psané v jazyce LEDA mohou být určeny jak pro mikropočítačové stanice přímého řízení technologického procesu - vázané těsně na jednotky styku s prostředím, tak i pro počítače nadřazeného řídicího systému, resp. pro počítače plnící převážně informační funkce operátorských stanic resp. dispečerských pracovišť.

Vedle prostředků specializovaných na problematiku přímého řízení, poskytuje jazyk LEDA prostředky běžné u univerzálních jazyků, zejména příkazy typu IF ELSE, WHILE, FOR, SWI, ... a je proto použitelný i pro řešení úloh charakteru vědeckotechnických a ekonomických výpočtů - např. simulačních, bilančních, statistických a optimalizačních úloh.

Jazyk LEDA se vyznačuje blokovou strukturou, umožňuje deklarovat proměnné s lokální platností a deklarovat uživatelské funkce volatelné jak globálně tak i s omezením na lokální volání.

Jazyk umožňuje zpracovávat proměnné velmi omezeného počtu standardních typů zaměřených na práci s technologickými veličinami, nepoužívá a ani neumožňuje deklarovat strukturované datové typy.

Konverze typu se provádí většinou automaticky a jen ve zvláštních případech, např. u některých parametrů funkcí, může typová kontrola vést k výskytu chyby hlášené překladačem.

Jazyk umožňuje pracovat vedle jednoduchých proměnných i s indexovanými proměnnými - jednodimenzionálními nebo vícedimenzionálními poli proměnných různých typů.

Vedle možnosti deklarace proměnných uživatelem nabízí každá konkrétní verze ZPV určitý soubor tzv. implicitních proměnných s pevně danými standardními identifikátory, které jednoznačně určují typ každé proměnné.

Část implicitních proměnných tvoří vstupní a výstupní pole umožňující styk uživatelského programu s okolním prostředím - tj. s D/A a A/D převodníky, s převodníky dvouhodnotových technologických signálů, ale také s jinými počítačovými stanicemi - jde o veličiny sdílené několika jednotkami řídicího systému, přičemž řízení vlastního přenosu dat mezi jednotkami řídicího subsystému (periodické aktualizace) může probíhat nezávisle na programu stanice realizující funkce zpracování technologických dat a generování akčních zásahů. Všechny implicitní proměnné mají globální platnost.

Program v jazyce LEDA může využívat rovněž tzv. systémové proměnné, které mají, stejně jako implicitní proměnné, pevně určené identifikátory a typy. Tyto proměnné slouží především pro plánování procesů v reálném čase a nesou informace o některých významných stavových veličinách stanice.

K efektivní práci na uživatelském programu přispívá možnost deklarace vlastních identifikátorů s mnemonikou odpovídající zvyklostem projektů MaR jak pro lokální proměnné, tak i jako alternativní označení implicitních proměnných.

Pro většinu průmyslových aplikací je podstatnou výhodou existence specializovaných standardních funkcí jazyka, které pokrývají všechny běžné požadavky na algoritmy přímého číslicového řízení a umožňují psát uživatelské programy velmi stručné a snadno udržovatelné. Využíváním standardních funkcí je také pozitivně ovlivňován výpočetní výkon mikropočítačových řídicích stanic.

Řada standardních funkcí jazyka má z hlediska zpracování signálů charakter dynamických subsystémů, jejichž vnější chování je závislé nejen na okamžitých hodnotách vstupních signálů, ale i na minulých hodnotě výstupního signálu a popř. i jiných - tzv. stavových - proměnných. Jedná se zejména o funkce PID(PID regulátor), PDC(PDC regulátor), FIL, CNT, MAV, MMA, MMI, TDL, SDR, DSR, IMP, DIF, MSE, tedy o "funkční bloky" regulátorů, filtru, čítače, zpracování signálů na "klouzavých intervalech" a časových logických funkcí. Tyto funkce jsou v jazyce LEDA užívány vlastně jako objekty, které mají vedle procedurální části také svou datovou část, která je obecně uživateli přístupná jen zprostředkovaně. Pro každé volání některé z těchto funkcí vzniká samostatný objekt s vlastním stavovým vektorem (jedinečnou vlastní historií) nulovaným při tzv. studeném startu příslušné počítačové stanice. Z tohoto důvodu nelze tyto "dynamické funkce" vyvolávat uvnitř příkazů cyklu s indexovanými vstupy či výstupy nebo v těle funkcí deklarovaných uživatelem.

Sortiment použitelných operátorů pokrývá běžné potřeby vykonávání aritmetických a logických operací na signálech a vyhodnocování relací - další požadavky lze splnit rozšiřováním knihovny standardních funkcí jazyka nebo deklarováním uživatelských funkcí. Aditivní a multiplikativní operátory nejsou v LEDA přetíženy určením pro aritmetické a zároveň i logické operace.

Pro jazyk LEDA je charakteristická možnost kombinovat ve zdrojovém textu příkazy určené k vykonávání v exekutivě řídicí stanice s příkazy vykonávanými v jiném počítači (PC) určeném pro styk s operátorem a s příkazy k získávání a zobrazování hodnot technologických proměnných (s využitím meziprocessorového spojení). Jazyk LEDA poskytuje prostředky pro zpracování částí vlastního zdrojového textu a jeho programově řízené modifikace, např. pro generaci periodicky zobrazovaných řetězců ASCII znaků odpovídajících momentálním hodnotám proměnných uživatelského programu nebo periodicky modifikovaných grafických reprezentací hodnot proměnných, resp. souborů hodnot proměnných, formou bodových grafů nebo sloupkových (bar) grafů.

Tato vlastnost jazyka umožňuje mj. také deklarovat uživatelsky formuláře pro zadávání parametrů a sledování skutečných hodnot vstupních a výstupních proměnných některých standardních funkcí (např. funkce PID), uživatelem deklarovaných funkcí nebo jiných - prakticky libovolných - částí aplikačního programu.

Je tedy možné vytvořit - prostředky jazyka - pro určitou oblast aplikací prakticky úplný soubor formulářů a další uživatelské programování může probíhat způsobem označovaným jako "Fill in Blank"; tento "formulářový" způsob programování je však pro jazyk LEDA jen zvláštním případem volného formátování komentovaných zdrojových textů aplikačních programů.

Jazyk LEDA poskytuje specializované prostředky pro práci s proměnnými, jejichž hodnoty jsou interpretovány jako fyzikální veličiny s obecně nelineárním vztahem k hodnotám vnitřní reprezentace těchto proměnných.

Syntaxe exekučních přiřazovacích příkazů vychází z konvence o směru šíření signálů ve schématu zleva doprava, přiřazení v obráceném smyslu je překladačem interpretováno jako příkaz pro dálkové akční zásahy resp. určení počátečních hodnot proměnných (seřizovaných parametrů). Jako symboly přiřazení resp. přenosu dat slouží znaky ">" a "<". Z hlediska teorie formálních jazyků je LEDA jazykem kontextovým.

Syntaxe i sémantika jazyka LEDA vykazuje některé rysy podobnosti s jazykem C, který slouží jako podpůrný pro programy překladače, programových komplexů pro vývoj programů v LEDA a interpretačních programů pro některé typy mikropočítačových řídicích jednotek. Je ovšem třeba konstatovat, že LEDA se vyznačuje ve srovnání s jazykem C omezenou mírou flexibility, která je podle názoru autora přiměřená pro řešení úloh uvažované aplikační třídy a vyplývá z ní nejen snazší zaškolení uživatelů, ale také vyšší stupeň tzv. zabezpečení jazyka. Na rozdíl od C je v LEDA zaveden typ Boolské proměnné resp. pole.

V LEDA se u standardních i uživatelem deklarovaných funkcí uplatňuje tzv. volání parametrů hodnotou (kopie v zásobníku) s automatickou dereferencí skutečných parametrů funkcí. Tato koncepce zajišťuje konzistenci obrazu prostředí vstupujícího do funkce, vylučuje předčasné předání rozpracovaných výstupů uživatelské funkce procesu s vyšší prioritou resp. jejich předčasný výstup ze stanice a umožňuje uživateli v jednotlivých voláních funkce na místě téhož formálního parametru použít jak identifikátor proměnné, tak i konstantu příslušného typu (např. i řetězcovou), a to bez potřeby jakéhokoli explicitního označování. Obecně může být skutečným parametrem funkce výraz.

Volání složených funkcí (skutečným parametrem funkce je volání funkce) nemá v LEDA obvyklou formu vnořeného volání, ale formu volání zřetěženého (zleva doprava), která přímo koresponduje se sériovým řazením funkčních bloků ve schématu. Forma zřetěženého volání funkcí je pro aplikační oblast zpracování signálů přirozenější, programy jsou čitelnější a LEDA přijetím této formy vychází vstříc požadavkům na usnadnění přechodu mezi blokovým schématem resp. signálovým diagramem a programem v textové formě.

## 3. Popis jazyka LEDA

### 3.1 Komentáře

Překladač jazyka LEDA (resp. jeho lexikální analyzátor) ignoruje ve zdrojovém textu s výjimkou tzv. "markerů" všechny řetězce znaků ohraničené zleva dvojicí znaků "/\*" a zprava dvojicí "\*/" (komentáře), dále také ignoruje identifikátor programu, identifikátory sítí a třinácté a další znaky identifikátorů proměnných, polí, funkcí a tzv. reprezentací (zvláštního druhu funkcí). Nevýznamné jsou (mimo lexikální jednotky jazyka) pro překlad také znaky: mezera, "CR", "LF", "FF", tabulátor a všechny další netisknutelné ASCII znaky.

Komentářové závorky "/\*" a "\*/" nemusí být použity v úvodní části programu - v oblasti tzv. ekvivalencí řetězců, tj. počínaje znakem ";" za klíčovým slovem CHAN až po klíčové slovo BEGIN (začátek první sítě v programu), mezi END sítě a BEGIN následující sítě a za END celého programu. Rovněž návěští mohou být (bez komentářových závorek) využity také - nebo jen - jako komentáře (dále podrobněji).

Překladač umožňuje používat vnořené komentáře, je tedy možné dočasně vyřadit část programu uzavřením mezi komentářové závorky, bez ohledu na to, že uvnitř závorek se nacházejí původní komentáře.

Z uvedeného vyplývá, že zdrojový text programu může být různými způsoby graficky upravován, např. se zdůrazněním vnořených struktur umístěním korespondujících dvojic příkazových závorek do stejného sloupce.

### 3.2 Proměnné

Každá proměnná používaná v programu musí být pevně zařazena do určité "paměťové třídy", a to v závislosti na tom, zda má pevně vyhrazenou určitou vlastní paměťovou oblast, nebo využívá dynamicky za běhu programu přidělovanou oblast (v zásobníku) - jedná se o proměnné statické, nebo jde o tzv. automatické proměnné s pouze dočasným významem. Další neměnnou vlastností každé proměnné je její typ, který určuje jednak velikost přidělené paměťové oblasti a dále také způsob manipulace s hodnotami proměnné - např. při vyhodnocování výrazů.

#### 3.2.1 Typy proměnných

Programy psané v jazyce LEDA mohou využívat proměnné těchto typů:

Typ	délka [bit]	rozsah	Popis
FIXP	16	<-32768, 32767>	číslo v pevné řádové čárce
FLTP	32	<-10 <sup>38</sup> , 10 <sup>38</sup> >	číslo v pohyblivé řádové čárce
BYTE	8	<0,255>	osmibitové číslo bez znaménka
BOOL	1	<0,1>	dvouhodnotová proměnná
CHAR	8	<0,255>	znak ASCII popř. semigrafický znak
String	-	-	řetězec znaků

Skupiny proměnných typu CHAR - uvozené číslem (typu BYTE) udávajícím délku - tvoří řetězce jako proměnné samostatného typu string. Tyto proměnné se deklarují bez užití speciálního klíčového slova typu (dále podrobněji ve zvláštní kapitole).

Proměnné typu FIXP mají rozsah <-32768, 32767>, přičemž "strojovou jednotkou" odpovídající normalizovaným maximálním hodnotám vstupních signálů z A/D převodníku je hodnota 8000 reprezentovaná standardně jako 100.0% - je tedy v této reprezentaci rozsah FIXP proměnných <-409.5%, 409.5%>.

Proměnné typu FLTP mají formát float čísel závislý na typu procesoru a překladače použitého pro překlad programů interpreteru. V současné době se jedná o formát odpovídající standardu IEEE-754 s 23bitovou normalizovanou mantisou.

Proměnné typu BYTE mají rozsah <0, 255>, "strojové jednotce" odpovídá číslo 250 reprezentované standardně jako 100.0% - maximum je tedy 102.0%.

### 3.2.2 Implicitní proměnné

Typ proměnné je určen pevně u tzv. implicitních proměnných se standardními identifikátory. Identifikátor implicitní proměnné je - až na dále popsané výjimky - tvořen písmenem, za kterým následují dvě nebo tři hexadecimální cifry, přičemž první písmeno je vlastně identifikátorem pole 256ti jednoduchých proměnných typu FIXP nebo FLTP, nebo identifikátorem pole 4096ti proměnných typu BOOL. Na každou (jednoduchou) implicitní proměnnou se proto může program odkazovat jako na jednoduchou proměnnou se standardním identifikátorem nebo jako na indexovanou proměnnou - prvek jednodimenzionálního pole. Identifikátorem pole implicitních proměnných může být samotné písmeno, nebo kompletní standardní identifikátor jednoduché implicitní proměnné.

Nulový index odkazuje na prvek pole, kterým je jednoduchá proměnná s identifikátorem totožným s identifikátorem pole tvořeným identifikátorem implicitní proměnné.

Indexová aritmetika zajišťuje při inkrementaci indexu posun skutečné adresy datového objektu v souladu s typem proměnných o 1, 2 nebo 4 byte. Jestliže index překročí hranice polí implicitních proměnných, pokračuje adresace se zachováním kroku podle původního typu.

Implicitní proměnné typu BOOL mají identifikátor tvořený písmenem a trojicí hexadecimálních cifer - první dvě určují relativní adresu slova (šestnáctibitového), třetí cifra určuje bit ve slově, a to tak, že číslice 0 odpovídá nejnižšímu binárnímu řádu, číslice F nejvyššímu.

Jestliže je identifikátor implicitní proměnné tvořen písmenem "F" a následují dvě hexadecimální cifry, jde o implicitní proměnnou typu FLTP.

Ostatní implicitní proměnné jsou typu FIXP s tím, že identifikátory začínající písmeny:

"A", "S", "X", "O", "Z"

označují proměnné tvořené šestnácticemi proměnných typu BOOL. Adresace jednotlivých bitů je povolena rovněž ve vstupních a výstupních polích; u proměnných typu FLTP zásadně možná není.

Všechny proměnné polí "S", "P", "W", "F"

jsou při tzv. studeném startu programu procesní stanice nastaveny na počáteční hodnoty určené konstantami vyskytujícími se v tzv. komunikačních přiřazeních; explicitně neinicializované proměnné jsou zároveň vynulovány.

#### 3.2.2.1 Vstupní a výstupní pole

Vedle implicitních proměnných typu FLTP a proměnných typu FIXP, z nichž některé jsou zároveň poli proměnných typu BOOL, existují pole implicitních proměnných typu BYTE adresovatelných jako indexované proměnné s indexy v rozsahu 0..255 nebo 0..511. Jedná se o proměnné:

1. Implicitní proměnné tvořící pole s identifikátorem IL - toto pole je určeno pro vstup dat z lokální sběrnice nebo ze vstupní strany tvořící přímo připojené periferní zařízení.
2. Implicitní proměnné tvořící pole s identifikátorem IR - toto pole je určeno pro vstup dat z terminálu.
3. Implicitní proměnné tvořící pole s identifikátorem OL - toto pole je určeno pro výstup dat na lokální sběrnici nebo na výstupní stranu tvořící přímo připojené periferní zařízení.
4. Implicitní proměnné tvořící pole s identifikátorem OR - toto pole je určeno pro výstup dat na terminálem.

Pole IL, OL, jsou tvořena 256ti, pole IR,OR 512ti proměnnými typu BYTE.

Jednotlivé proměnné polí 1),...,4) mohou však být odkazovány rovněž identifikátory tvořenými identifikátorem pole s bezprostředně následujícím hexadecimálním indexem - bez hranatých závorek. Je-li zadána třetí cifra v rozsahu 0 ... 7, vzniká identifikátor určité BOOL proměnné daného byte.

Adresace jednotlivých byte ve vícebytových údajích a posloupnost BOOL proměnných v rámci FIXP proměnné se liší podle implementace interpreterů jazyka LEDA na různé typy řídicích stanic resp. v souladu s konvencemi o adresaci byte dodržovanými různými překladači jazyka C (konkrétní řešení je součástí popisu implementace jazyka LEDA).

Proměnné ze vstupních polí jsou zpravidla zpracovávány pomocí standardní funkce IPR, do výstupních polí jsou data zpravidla ukládána pomocí standardní funkce OPR, která zároveň zajišťuje převod signálů do formátu požadovaného v navazujících zařízeních.

Vstupní a výstupní pole nemusí být využívána ve všech implementacích - např. u systémů PEP lze určit jako vstupní a výstupní libovolné deklarované proměnné typu BOOL, BYTE, FIXP nebo FLTP.

**Implicitní proměnné mají globální platnost - jejich "oblastí viditelnosti" je celý program.**

### 3.2.2.2 Specifikace polí implicitních proměnných

Pole implicitních proměnných, která mají být skutečně uživatelskému programu dostupná se určí zapsáním jejich identifikátorů - vždy jen jediného písmene - do seznamu polí za klíčové slovo ARRAY uvedené za identifikátor programu, tedy na začátku zdrojového textu celého programu. Tato specifikace polí implicitních proměnných je povinnou součástí každého programu napsaného v jazyce LEDA. Vstupní a výstupní pole jsou povinnou součástí polí implicitních proměnných a ve specifikaci se neuvádějí. Jazyk LEDA obecně umožňuje specifikovat až 26 dalších polí s identifikátory tvořenými všemi písmeny anglické abecedy.

### 3.2.3 Systémové proměnné

Program v jazyce LEDA může využívat vedle implicitních proměnných několik dalších uživatelem nedeklarovaných proměnných s globální platností a významem pro část programu zpracovávanou určitým procesorem. Jedná se o následujících patnáct systémových proměnných určených pro řízení operací v reálném čase, dále o proměnnou pro odměřování krátkých časových intervalů (exekutiv) a dvě BOOL proměnné:

Identifikátor	Typ	Význam
YEAR	BYTE	Letopočet - 1900
MONT	BYTE	Měsíc v roce (1 až 12)
DAY	BYTE	Den v měsíci (1 až 31)
HOURL	BYTE	Hodina dne (0 až 23)
HMIN	BYTE	Minuta v hodině (0 až 59)
MSEC	BYTE	Sekunda v minutě (0 až 59)
DMIN	FIXP	Minuta dne (0 až 1439)
YDAY	FIXP	Den v roce (1 až 366)
HSEC	FIXP	Sekunda v hodině (0 až 3599)
WDAY	BYTE	Pořadí dne v týdnu (0 až 6)
FMONT	BYTE	Měsíc posledního startu
FDAY	BYTE	Den posledního startu
FHOURL	BYTE	Hodina posledního startu
FMIN	BYTE	Minuta posledního startu
FMSEC	BYTE	Sekunda posledního startu
TIMEX	FIXP	Inkrementovaný čítač reálného času v ms
START	BOOL	Impuls s významem "studný start stanice probíhá"
RESTART	BOOL	Impuls s významem "teplý restart stanice probíhá"

**Trojice proměnných YEAR, MONT a DAY tvoří pole typu BYTE s identifikátorem SDATE a trojice proměnných HOURL, HMIN a SEC tvoří další systémové pole téhož typu s identifikátorem STIME.** S údajem systémového data SDATE lze pracovat jako s celkem pomocí standardní reprezentace DATE; s údajem systémového času STIME lze pracovat prostřednictvím standardní reprezentace TIME (identifikátory reprezentací DATE resp. TIME mohou tvořit skutečné parametry např. ve volání standardní reprezentace SCD, která potom zajistí zobrazení aktuálního data resp. časového údaje).

Proměnné FMONT, ..., FMSEC jsou plněny kopiemi hodnot proměnných MONT, ..., MSEC, a to vždy po skončení periody zpracování dat (všech periodických sítí aplikačního programu). Čas posledního startu je nutné vyhodnotit podmíněně (pomocí IF(START) nebo IF(RESTART)).

Aplikační program psaný v jazyce LEDA může popisovat algoritmus vykonávaný paralelně několika procesory a v paměti každého z nich se průběžně aktualizují vlastní proměnné reálného času. V programu se mohou vyskytovat tzv. rozšířené identifikátory odkazující na externí proměnné obecně a tedy i externí proměnné systémového času, nebo identifikátory v té podobě, jak jsou zde uvedeny, odkazující alternativně na systémové proměnné ležící v paměti osobního počítače (terminálu operátora) jsou-li použity v tzv. MASTER síti, anebo na proměnné podřízené řídicí stanice s jejímž programem pracujeme prostřednictvím komunikační linky, jsou-li použity v libovolné síti interpretované v této řídicí stanici. Proměnné reálného času se aktualizují v každé periodě zpracování dat před spuštěním interpretace exekutivy aplikačního programu dané stanice. Z toho vyplývá, že tyto buňky lze prostředky aplikačního programu pouze dočasně modifikovat, nelze je však nastavit na nové počáteční hodnoty. Seřízení systémového času operátorské stanice se provádí prostředky operačního systému MS DOS.

Podrobný popis chování řídicích stanic při různých typech restartů je předmětem popisu implementace jazyka LEDA na konkrétní řídicí systémy.

### 3.2.4 Proměnné deklarované v uživatelském programu

Není-li proměnná implicitní nebo systémová, musí být v uživatelském programu deklarována s udáním typu, a to jako:

1. Lokální proměnná síť
2. Lokální proměnná bloku (bloku příkazů)
3. Lokální proměnná funkce (funkce deklarované uživatelem)
4. Lokální proměnná reprezentace

Je-li deklarace skupiny proměnných uvozena klíčovým slovem typu (FIXP, FLTP,...), jedná se o "dynamické" proměnné, které mají vyhrazeno místo v paměti jen krátkodobě, do opuštění příslušné sítě, bloku, funkce nebo reprezentace. Proměnné této třídy nelze inicializovat, ani jim nelze "komunikačně přiřazovat", tj. seřizovat jejich hodnoty.

Je-li v deklaraci proměnných před klíčové slovo typu předřazeno klíčové slovo STAT, jedná se o deklaraci statických proměnných lokálních v dané struktuře. Statické proměnné mají určitou oblast paměti vyhrazenou trvale. Všechny statické proměnné tzv. dynamické funkce mají trvale vyhrazen paměťový prostor pro všechna vyvolání funkce. Statické proměnné umožňují uchovávat stavové vektory dynamických funkcí, tedy údaje přenášené z jednoho výpočetního cyklu do následujícího.

Při studeném startu programu procesní stanice jsou všechny deklarované statické proměnné - stejně jako stavové proměnné dynamických funkcí - vynulovány. Deklarované statické proměnné proto nemohou být v procesní stanici explicitně inicializovány, ale lze jim "komunikačně přiřazovat".

Deklarace lokálních proměnných jsou vždy uvedeny na začátku sítě resp. bloku, pro který mají svou platnost. Při totožnosti identifikátorů deklarovaných v nadřazeném a podřízeném (vnořeném) bloku, popř. při hroící kolizi s globální (implicitní) proměnnou má zásadně při použití prioritu lokální proměnná - totéž platí i pro identifikátory funkcí.

Z vnořených bloků je možné se odkazovat na všechny proměnné deklarované v nadřazených blocích resp. v nadřazené síti.

#### 3.2.4.1 Deklarace proměnných

Deklarace proměnných je vždy uvozena klíčovým slovem typu proměnné (FIXP, FLTP, BOOL, BYTE, ...), nepovinně se předřazuje klíčové slovo STAT pro staticky alokované proměnné a dále seznam identifikátorů deklarovaných proměnných (jednoduchých i polí) oddělených čárkami. Seznam je ukončen středníkem.

V rámci deklarace je povolena také inicializace proměnné - zapisuje se podobně jako tzv. skalární parametr - identifikátor proměnné je následován operátorem komunikačního přiřazení "<" za nímž následuje nepovinně identifikátor tzv. reprezentace ukončený znakem "\$" a dále konstanta příslušného typu (číselná konstanta nebo řetězcová konstanta), smysl přiřazení ani hodnoty proměnných však nelze za běhu měnit. Inicializovat lze výhradně statické proměnné.

### 3.2.4.2 Deklarovaná pole

Uživatelský program se může odkazovat na indexované proměnné tvořící pole deklarovaná v programu. Pole může být deklarováno s udáním libovolného typu jako statické (předřadí se klíčové slovo STAT) nebo jako dynamické (dočasně významu). Za klíčovým slovem typu následuje klíčové slovo ARRAY. Počet dimenzí deklarovaného pole určuje počet dvojic hranatých závorek zapsaných za identifikátor pole; uvnitř závorek je celé číslo udávající počáteční hodnotu indexu - chybí-li, je automaticky nulová, následuje znak "#" a číslo udávající počet prvků pole pro daný index. Alternativně lze v hranatých závorkách zapisovat dvojice čísel oddělených dvěma tečkami ".." a udávajících spodní a horní mezní hodnotu indexu. Správná je tedy např. deklarace:

```
STAT FLTP ARRAY ParamA[0..8] [1..3];          /* dvourozmerne pole 27mi prvku*/
```

V deklaraci pole lze inicializovat jeho prvky - místo jediné konstanty jako u jednoduché proměnné zadáme skupinu konstant oddělených mezerami, a to v pořadí nárůstu indexů. Nejrychleji se mění poslední index vícedimenzionálního pole.

### 3.2.4.3 Ekvivalence alokace polí

Uživatelem deklarovaná statická pole mohou zaujímat jinak nevyužitou oblast paměti proměnných, alternativně však může být při deklaraci určeno, že deklarovaná pole budou zaujímat oblast paměti počínaje již dříve (v programu výše) deklarovanou proměnnou nebo polem deklarovaných či implicitních proměnných. Různá pole různých typů se tedy mohou vzájemně překrývat, tzn., že určité paměťové lokace mohou být z aplikačního programu přístupné jako proměnné různých typů.

Má-li být deklarované pole ekvivalentní s jiným polem nebo jeho částí, je nutné za poslední uzavírací hranatou závorku v deklaraci doplnit klíčové slovo EQU následované identifikátorem výše deklarované proměnné, jejíž adresa určí adresu prvního prvku deklarovaného pole. Např. deklarace:

```
STAT CHAR ARRAY PoleCh[1..5] EQU P4A;
```

zavádí statické znakové pole PoleCh s pěti prvky umístěnými překladačem do oblasti pole implicitních proměnných P. V tomto případě může být výhodné, že PoleCh lze inicializovat (komunikačním přiřazením) a přitom je díky vlastnostem pole P zajištěna jeho nová inicializace při každém studeném startu aplikačního programu.

### 3.2.5 Identifikátory

Identifikátor proměnné nebo pole je tvořen řetězcem povolených znaků (písmen, číslic a podtržíték), z nichž první musí být písmeno.

Překladač LEDA rozlišuje až dvanáctiznakové identifikátory, ale delší identifikátory nepovažuje za syntakticky chybné. Toho lze využívat pro zlepšení čitelnosti zdrojového textu. Překladač rozlišuje velká a malá písmena.

Identifikátory proměnných nesmějí být totožné s žádným z tzv. rezervovaných slov, tj. klíčových slov jazyka a identifikátorů standardních funkcí nebo standardních reprezentací. Klíčová slova se píší velkými písmeny, identifikátory standardních funkcí a reprezentací rovněž.

Stejná pravidla jako pro identifikátory proměnných a polí platí i pro identifikátory funkcí, reprezentací a pro návěští, s tím, že je povoleno deklarovat funkce nebo reprezentace s identifikátorem totožným s identifikátorem některé standardní funkce nebo reprezentace a v dané oblasti programu (globálně nebo v některém bloku) tak překrýt platnost standardní funkce či reprezentace.

### 3.2.6 Konstanty

Konstanty jsou dvojího typu:

1. řetězcové konstanty,
2. číselné konstanty.

Řetězcová konstanta může být vstupním parametrem některých standardních funkcí nebo vstupním parametrem reprezentace - tj. funkce pracující mimo exekutivu výkonné stanice pro kterou je program určen a může se vyskytovat ve výrazech.

Řetězcová konstanta je tvořena skupinou ASCII znaků uzavřených mezi uvozovky. Netisknutelné znaky lze nahradit trojicí znaků, z nichž první je "#" a další dva udávají hexadecimálně kód netisknutelného znaku. Tuto trojici je ovšem možné pro lepší čitelnost textu nahradit s využitím ekvivalence řetězců používanou zkratkou, např. LF pro přechod na nový řádek. Do řetězcových konstant lze zapsat mj. různé řídicí sekvence (začínající např. znakem CTRL), které mohou sloužit pro změnu typu písma nebo řízení tiskárny v grafickém režimu.

Číselná konstanta může být vstupním parametrem standardních nebo uživatelských funkcí nebo reprezentací a může vystupovat ve výrazech. V některých případech může být číselná konstanta také výstupním parametrem standardní funkce (s významem logického čísla výstupního kanálu).

Číselná konstanta je tvořena číslem, které může být následováno řetězcem znaků neobsahujícím klíčová slova (zpravidla jde o symbol fyzikální jednotky) a dále označením reprezentace, která zajistí převod celého řetězce do vnitřní reprezentace odpovídající některému z typů proměnných (FIXP, FLTP, BYTE).

Není-li typ číselné konstanty určen typem vstupu použité reprezentace - reprezentace není použita vůbec nebo jde o standardní reprezentaci (např. \$D), jejíž vstupní typ není pevně stanoven - určuje typ konstanty forma jejího zápisu jako:

- celého čísla bez znaménka nebo se znaménkem - typ FIXP,
- čísla desetinného (s des. tečkou), popř. v exponenciálním tvar (mantisa, znak "E", exponent) - typ FLTP.

Konstanta ať již řetězcová nebo číselná, je součástí speciálního návěští v příkazu SWI.

V binárním produktu překladu se stejně jako číselná konstanta projeví výsledek vyhodnocení tzv. konstantního výrazu. V konstantním výrazu nevystupují proměnné, jejichž hodnoty nelze v době překladu určit.

### 3.2.7 Reprezentace

Reprezentace je zvláštním druhem funkce vykonávané v nadřazeném počítači určeném pro styk s operátorem, a to jen v případě:

1. že je umístění vstupních nebo výstupních řetězců reprezentace součástí zdrojového textu programu ve „výřezu“ zobrazeném v dané chvíli programem TOLEDA na monitoru;
2. při překladu zdrojového textu LEDA, kdy jsou všechny reprezentace vyskytující se ve zdrojovém textu spuštěny tak, aby byly získány hodnoty proměnných programu pro jejich inicializaci (počáteční hodnoty některých proměnných jsou součástí souborů .BED a .SED).

Některé reprezentace mohou sloužit pro převod konstant zadávaných operátorem na hodnoty vnitřní reprezentace proměnných, k jejichž identifikátorům směřuje přiřazovací znaménko "<" nebo ">". Reprezentace mohou zpracovávat jak číselné tak také řetězcové konstanty.

Jiné reprezentace slouží pro převod hodnot vnitřní reprezentace proměnných na řetězce znaků zobrazovaných do zdrojového textu na místo nejbližšího výskytu znaku nebo skupiny znaků "\_" (podtržítka) po identifikátoru reprezentace. Reprezentace mohou poskytovat také grafický výstup.

Umístění vstupních resp. výstupních řetězců může být určeno rovněž nepřímo tzv. odkazem. Na místo vstupního řetězce resp. skupiny podtržitek poblíž identifikátoru reprezentace se zapíše řetězec znaků tvořící tzv. marker, jehož nezbytnou součástí je znak "\". Skutečné umístění řetězce zpracovávaného reprezentací resp. místo zobrazení řetězce generovaného reprezentací určuje místo nejbližšího výskytu markeru ve zdrojovém textu, a to počínaje místem vyvolání reprezentace. Určitý marker tedy může být v programu použit vícenásobně. Pro "zadávací" směr reprezentací určují polohy markerů v operátorské obrazovce začátky tzv. zadávacích okének, která jsou obsluhována v programovém prostředí TOLEDA. Šířka každého zadávacího okénka je určena typem proměnné, jejíž hodnota má být modifikována operátorem za běhu TOLEDA. Pro typ BYTE mají okénka šířku tří znaků, pro typ FIXP šířku osmi znaků a pro typ FLTP šířku deseti znaků. Různá zadávací okénka se nemají navzájem překrývat.

Markerem smí být libovolný identifikátor uvozený znakem "\"; ve volání reprezentace je marker doplněn zprava dalším znakem "\".

Tentýž marker lze s různými posuny počátku řetězců použít v několika voláních různých reprezentací.

Nepřímým určením místa zobrazení pomocí "odkazů" lze zadávat tzv. "dynamiku" mnemotechnických schémat, formátovat zobrazované údaje do tabulek, údaje zadávané operátorem vřadit do komentářů (obecně proměnných) a vytvářet tak uživatelsky zadávací formuláře nebo programovat dialogový systém komunikace s operátorem.

Jako příklad uveďme použití markerů: \M6, \A70 a několik vyvolání reprezentací \$TT a \$T2:

```
A > TT\M6\; /* marker "M6\" bez posunu */
X5A > T2$\A70\+4; /* marker "A70\" s posunem +4 */
\M5\ -6 $T2 < a; /* marker "\M5\" s posunem -6 */
REP a < < \M6\; /* zadání číselné hodnoty na markeru \M6 */
REP a < $TT < \M6\; /* totéž přes standardní reprezentaci TT */
```

Vyhrazený marker \HOME označuje levý horní roh obrazovky bez ohledu na momentálně zobrazovaný výřez zdrojového textu. Odkazem \HOME\ (příp. se zadaným posunem lze zajistit např. podmíněné zobrazování výstražných hlášení generovaných v periodických sítích interpretovaných v operátorské stanici (interpret XEDA1). Marker \HOME se ve zdrojovém textu vyskytuje jen jako parametr ve volání reprezentací - poloha pro zobrazování nebo čtení operátorem zadávaných řetězců je tímto markerem určena implicitně. Odkaz na \HOME se záporným posunem směřuje do plochy obrazovky počínaje posledním sloupcem posledního řádku (pro posun -1) a dále pro záporná posunutí s rostoucí absolutní hodnotou odkazuje na znakové pozice ležící vlevo od pravého dolního rohu obrazovky a v řádcích ležících výše.

Překladač jazyka LEDA zpracovává dále také tzv. speciální markery tvořené znakem "\" následovaným dekadickým číslem kódu klávesy standardní klávesnice PC. Speciální marker označuje řádek (může ležet v kterémkoli jeho sloupci), který se má po stisknutí klávesy s daným kódem stát prvním řádkem zobrazovaného výřezu textu programu; předpokládá se přitom, že program byl spuštěn v prostředí COLEDA (je v režimu Run). Např. speciální marker \315 koresponduje s klíčem F1, marker \316 s klíčem F2, ... (skutečné kódy lze pro jednotlivé klávesy zjistit pomocí systémové proměnné LKEY).

Označení reprezentace je tvořeno identifikátorem reprezentace a znakem "\$", který bezprostředně předchází nebo následuje identifikátor reprezentace. Podrobně jsou pravidla zápisu reprezentací podchycena syntaktickými diagramy jazyka LEDA, v zásadě však lze pravidlo zápisu označení reprezentace ve skalárních parametrech formulovat takto: znak \$ stojí na opačné straně identifikátoru reprezentace proti symbolu operátoru komunikačního přiřazení "<" nebo ">", přičemž tyto operátory jsou umístovány blíž k identifikátoru proměnné, jejíž hodnota má být zobrazována, anebo jí má být komunikačně nějaká hodnota přiřazována. Je tedy často identifikátor reprezentace uzavřen mezi operátor komunikačního přiřazení a symbol "\$".

Reprezentace může, stejně jako funkce, zpracovávat více než jeden vstupní parametr a rovněž výstupních parametrů resp. zobrazovaných hodnot může být větší počet. Reprezentace může být také dynamická, tzn., že může mít přiřazenu staticky oblast paměti, kterou používá pro uchovávání svého stavového vektoru (záznamu historie).

Vstupním parametrem reprezentace může být výraz. Často je vstupním parametrem reprezentace číselná konstanta nebo odkaz na řetězec znaků číselné konstanty.

Je-li znak "\$" uveden, ale identifikátor reprezentace je prázdný, jedná se o vyvolání implicitní reprezentace, kterou určuje překladač podle kontextu - pro jednotlivé vstupní a výstupní parametry standardních funkcí pevně, u uživatelských funkcí v souladu s jejich deklarací.

Všechny konstanty ve výrazech a skutečných parametrech volání standardních nebo uživatelských funkcí mohou tedy být zadávány a zobrazovány:

- bez označení reprezentace (bez identifikátoru reprezentace i bez znaku "\$")
- prostřednictvím reprezentace implicitní pro určité místo použití
- prostřednictvím reprezentace s udaným identifikátorem reprezentace.

V případě 1. jsou automaticky užívány standardní reprezentace, které přiřazují hodnotám proměnných typu BOOL znaky "0" nebo "1", proměnným typu BYTE, FIXP a FLTP příslušné řetězce číslic doplněné pro FLTP znakem znaménka, desetinné tečky, příp. symbolem "E" uvozujícím exponentovou část.

V případě 2. jsou mimo parametry funkcí užívány pro proměnné FIXP a BYTE standardní reprezentace v % rozsahu. Hodnoty proměnných FIXP jsou v tomto případě zobrazovány a zpracovávány jako číselné řetězce doplněné znaménkem a znakem "%". Rozsah těchto hodnot je "-409.5%" až " 409.5%", přičemž údaj "100.0%" odpovídá hodnota 8000. Proměnné typu BYTE mají rozsah 0% až 102%, což odpovídá hodnotám vnitřní reprezentace v rozsahu 0 až 255.

Závěrem tohoto odstavce je vhodné upozornit, že interpretace jednotlivých reprezentací nemá přímou souvislost s vykonáváním exekutivy tzv. MASTER sítí v PC (a tím méně exekutivy podřízených procesních stanic) - spouštění reprezentací nelze programově řídit! Není tedy možné např. podmíněně spouštění různých reprezentací pomocí větvení IF ... ELSE, pomocí příkazu SWI ... . Možné ovšem je mj. i podmíněně určovat vstupní parametry reprezentací v exekutivě některé MASTER sítě, nebo i sítě procesní stanice.

### ***3.2.7.1 Standardní reprezentace***

Soubor standardních reprezentací konkrétní verze základního programového vybavení může obsahovat reprezentace podle následujícího seznamu.

Identifikátor	reprezentace (poznámka)
<u>\$X</u>	hexadecimální reprezentace pro proměnné typu BYTE
<u>\$XX</u>	hexadecimální reprezentace pro proměnné typu FIXP
<u>\$B</u>	binární reprezentace pro proměnné typu BYTE
<u>\$BB</u>	binární reprezentace pro proměnné typu FIXP
<u>\$D</u>	dekadická pro proměnné FIXP, FLTP i BYTE (vč.barvy)
<u>\$CD</u>	obdobu \$D s řízenou barvou pozadí a číslic
<u>\$TEXT</u>	reprezentace proměnné typu BOOL (alternativní textové řetězce - např. "ON"/"OFF")
<u>\$MTEXT</u>	reprezentace skupiny proměnných typu BOOL různými textovými řetězci
<u>\$GTEXT</u>	index určuje řetězec z pole řetězců a barvu pozadí a znaků z tzv. palety barev
<u>\$TAB</u>	digitální reprezentace skupiny proměnných FIXP nebo FLTP nebo BYTE (tabulka)
<u>\$BAR</u>	vertikální nebo horizontální sloupkový graf
<u>\$TIGR</u>	grafy časových závislostí (s lineární interpolací)
<u>\$TICUR</u>	grafy čas.závislostí pro vzorky z archivních souborů
<u>\$XYCUR</u>	grafy v souřadnicích x,y
<u>\$DATE</u>	práce s datem ve tvaru RR-MM-DD, např.: 92-05-31
<u>\$TIME</u>	práce s časovým údajem hh:mm:ss, např.: 23:59:59
<u>\$BY</u>	pro proměnné typu BYTE jako číslo 000 až 255
<u>\$FI</u>	pro proměnné typu FIXP jako číslo -32768 až 32767
<u>\$FL</u>	pro proměnné typu FLTP
<u>\$BP</u>	pro proměnné typu BYTE v % "strojové jednotky"
<u>\$FP</u>	pro proměnné typu FIXP v % "strojové jednotky"
<u>\$FSmn</u>	číslo se znaménkem, m číslic před des.tečkou, n za
<u>\$FUmn</u>	číslo bez znaménka, m číslic před des.tečkou, n za
<u>\$F1K</u>	průtok 0 až 1000.0 m3/hod
<u>\$FK4</u>	průtok 0 až 400.0 m3/hod
<u>\$FK25</u>	průtok 0 až 250.0 m3/hod
<u>\$FK16</u>	průtok 0 až 160.0 m3/hod
<u>\$P4M</u>	tlak -4.000 až 4.000 MPa
<u>\$P1M6</u>	tlak -1.600 až 1.600 MPa
<u>\$T2</u>	teplota -050.0 až 200.0 °C
<u>\$T5</u>	teplota 000.0 až 500.0 °C
<u>\$TT</u>	teplota -999.9 až 999.9 °C
<u>\$PP</u>	tlak -9.999 až 9.999 MPa
<u>\$FF</u>	průtok obj. -999.9 až 999.9 m3/h
<u>\$MM</u>	průtok hm. -999.9 až 999.9 t/h
<u>\$EE</u>	energie -999.9 až 999.9 GJ
<u>\$WW</u>	výkon -99.99 až 99.99 MW
<u>\$LL</u>	hladina -999.9 až 999.9 cm
<u>\$QQ</u>	objem -999.9 až 999.9 m3
<u>\$GG</u>	hmotnost -999.9 až 999.9 t

Další standardní reprezentace které nemají vlastní uživatelsky dostupný identifikátor slouží pro zobrazování a zadávání některých parametrů standardních funkcí - jde o výlučně implicitní reprezentace určené pouze místem použití.

### 3.2.8 Operátory

V jazyce LEDA je povoleno použití následujících operátorů:

Priorita	
	<b>Operátory žádostí o zobrazení hodnot resp. stavů proměnných:</b>
1	> vlevo je proměnná, vpravo skupina znaků "_" eventuálně s předřazeným identifikátorem reprezentace, který je následován znakem "\$", popř. je skupina znaků "_" nahrazena odkazem určujícím místo zobrazení
1	< vpravo je proměnná, vlevo skupina znaků "_" následovaná eventuálně znakem "\$" a identifikátorem reprezentace, popř. je skupina znaků "_" nahrazena odkazem
	<b>Operátor komunikačního přiřazení:</b>
1	< vlevo je proměnná, vpravo je konstantní výraz nebo odkaz, případně s předřazeným identifikátorem reprezentace následovaným znakem "\$"
	<b>Unární operátory:</b>
2	! negace dvouhodnotové proměnné (výrazu)
2	- inverze analogové proměnné (výrazu)
	<b>Binární operátory:</b>
4	* součin
4	& konjunkce (AND)
4	/ dělení
5	+ součet nebo sjednocení řetězců
5	- rozdíl
5	disjunkce (OR)
	<b>Relační operátory:</b>
6	<< menší než (ostrá nerovnost)
6	>> větší než (ostrá nerovnost)
6	<= menší nebo rovno (implikace pro dvouh.)
6	>= větší nebo rovno
6	== rovnost (ekvivalence dvouhodnotových)
6	!= nerovnost (nonekvivalence dvouhodnotových)
	<b>Operátor exekučního přiřazení:</b>
7	> vlevo je proměnná nebo výraz nebo seznam vstupních parametrů funkce nebo identifikátor funkce, vpravo je proměnná nebo seznam proměnných či výstupních parametrů funkce nebo identifikátor funkce.

Priorita operátorů klesá v uvedeném pořadí.

### 3.2.9 Výrazy

Výrazy se tvoří z operandů, operátorů a kulatých závorek, které se vždy musí vyskytovat ve dvojici.

Operátory povolené v LEDA již byly uvedeny; operandy mohou být:

- (skalární) proměnné, a to jednoduché nebo indexované (prvky polí resp. matic),
- konstanty (číselné nebo řetězcové), popř. odkazy na konstanty
- funkce (volání funkcí),
- (skalární) parametry.

Skalární parametr je skalární proměnná, která má bezprostředně zprava nebo zleva popř. z obou stran zapsány operátory žádostí o zobrazování skutečných hodnot ">", "<" popř. zprava operátor "<" komunikačního přiřazení a dále směrem vlevo nebo vpravo následuje (nepovinně) označení reprezentací a konstanty nebo skupiny znaků "\_" (podtržítka), popř. "odkazy".

Skalární parametr má ve výrazu stejné postavení jako proměnná, ale zároveň může být v rámci skalárního parametru přiřazena proměnné přes zvolenou reprezentaci počáteční hodnota; skutečná hodnota proměnné (získávaná

průběžně dotazovými zprávami ze vzdálené jednotky) může být přes zvolenou reprezentaci zobrazována, popř. lze v rámci parametru dát operátorovi možnost v libovolném okamžiku modifikovat hodnotu proměnné.

## Příklady

Syntakticky správné jsou např. tyto skalární parametry:

```

_____ $TT < Teplota18 ; /* zobrazovaná FIXP proměnná */
Teplota18 > TT$ _____ ; /* zobrazovaná FIXP proměnná */
Teplota18 < TT$ 128.5'C ; /* seřiditelná FIXP proměnná */
_____ < Teplota18 < TT$ 128.5'C ; /* seřiditelná FIXP proměnná se souč. zobrazením v bitech */
_____ $ < Teplota18 > TT$ _____ ; /* FIXP proměnná zobraz. v % rozsahu a zároveň ve 'C */
Teplota18 > _____ > TT$ _____ ; /* FIXP proměnná zobraz. v % rozsahu a zároveň ve 'C */
Pole1[26] > _____ ; /* zobrazovaný prvek pole libovolného typu (v bitech) */
Pole1[26] < 32767 ; /* seřizovatelný prvek pole libovol. typu (v bitech) */
_ < RezimAUT ; /* zobrazovaná BOOL proměnná */
RezimAUT > _ ; /* zobrazovaná BOOL proměnná */
RezimAUT < 1 ; /* modifikovatelná BOOL proměnná */

```

Následují příklady užití skalárních parametrů ve výrazech:

```

X00-4*X01 > X02 > _____ ; /* Zobrazovány výsledek aritmetického výrazu */
( _____ < X00 ) - 4 * X01 > X02 > _____ ; /* Zobrazena navíc vstupní proměnná */
(X00 < -515) - 4 * X01 > X02 > _____ ; /* Seřiditelná vstupní proměnná */
(S00 < -515) - 4 * X01 > X02 > _____ ; /* Seřiditelná vstup. proměnná se zadanou inic. hodnotou */
_____ < U1, U2 > _____ > MAX > Maximum > _____ ; /* Zobrazování vstup. param. funkce */
_____ < U1, U2 < -12830 > MAX > Maximum > _____ ; /* Zobrazování a modifikace hodnot vstup. parametru funkce */
IF ((RezimNajizdeni > _)) { 1 > Vent1ON; 1 > Vent2ON;
                            1 > Cerp1ON; 1 > Cerp2ON;
                            0 > RezimNajizdeni
                            };

```

**Jednotlivé operace se při vyhodnocování výrazu provádějí v pořadí určeném:**

- uzávorkováním - vnitřní závorky mají prioritu,
- prioritou operátorů,
- zleva doprava neurčí-li prioritu ani 1. ani 2..

Tzv. faktor je tvořen skalární proměnnou nebo výrazem uzavřeným do kulatých závorek nebo voláním funkce či faktorem s předřazeným unárním operátorem "-" pro aritmetický faktor, "!" pro faktor typu **BOOL**.

Aritmetické faktory typu **BYTE**, **FIXP**, **FLTP** lze sdružovat do tzv. termů pomocí operátorů "\*" nebo "/" a termy do jednoduchých aritmetických výrazů pomocí operátorů "+" nebo "-". Dvojice jednoduchých výrazů může být relačním znaménkem "==" nebo "!=" nebo "<=" nebo ">=" nebo ">>" nebo "<<" spojena do výrazu typu **BOOL**.

Dvouhodnotové faktory lze sdružovat do termů pomocí operátorů "&" a termy do jednoduchých výrazů typu BOOL pomocí operátorů "|" nebo "||". Dvojice jednoduchých výrazů typu BOOL může být relačním znaménkem "==" nebo "!=" nebo "<=" spojena do výrazu typu BOOL.

Proměnné typu FIXP resp. BYTE chápané jako skupiny proměnných typu BOOL lze kombinovat také pomocí operátorů "&", "|", "||" s významem bitových operací konjunkce, disjunkce, exclusiv or, tedy skupiny šestnácti resp. osmi logických operací, které se provádějí mezi nultými bity, prvními bity, ... obou operandů. Obdobně je povoleno negovat všechny bity aritmetické proměnné FIXP resp. BYTE pomocí operátoru "!".

Dvouhodnotové proměnné, faktory, termy a jednoduché výrazy mohou být vzájemně kombinovány - uplatní se přitom automatické konverze typů popsané podrobně v dalším textu - avšak s těmito omezeními:

- v termu tvořeném smíšeně aritmetickými a BOOL proměnnými smějí být použity mezi veličinami těchto dvou kategorií jen operátory "\*" (aritmetický součin aritmetické a dvouhodnotové proměnné má smysl - dělení nebo konjunkce nikoli).
- v jednoduchém výrazu smí být mezi aritmetickou a dvouhodnotovou proměnnou použity jen operátory "+" nebo "-" (operátory "|" nebo "||" nemají v tomto případě smysl).

Při vyhodnocování výrazů se automaticky provádí konverze typů proměnných a konstant - výsledek vyhodnocení aritmetického výrazu má typ totožný s "nejdelším" vyskytujícím se typem proměnné nebo konstanty; relace se vyhodnocují vždy v "kratším" z obou typů. Obsahuje-li výraz více než jednu proměnnou nebo konstantu, vyhodnocuje se však vždy alespoň s typem FIXP - to znamená, že požadované aritmetické operace mezi proměnnými a konstantami typu BYTE probíhají až po konverzi nejméně na typ FIXP. Typy číselných konstant určuje překladač podle výskytu desetinné tečky (FLTP), avšak vždy nejméně jako typ FIXP.

Konstantní výrazy jsou výrazy, které neobsahují žádné proměnné a jsou proto vyhodnocovány již při překladu.

Skutečnými parametry funkcí jsou, až na výjimečné případy kdy smí být použita jen konstanta, výrazy, které mohou opět obsahovat volání funkcí. Závorky se ve výrazech tvořících parametry funkcí kumulují doleva, tzn., že např. výraz:

$$a + b > DIF > c;$$

dává stejné výsledky jako:

$$(a + b) > DIF > c;$$

Výraz:

$$a > IMP | b > IMP > c;$$

je ekvivalentní výrazu:

$$(a > IMP | b) > IMP > c;$$

a zcela něco jiného je:

$$(a > IMP) | (b > IMP) > c;$$

### 3.2.10 Příkazy

Vlastní zpracování dat včetně větvení programu se v jazyce LEDA - stejně jako u ostatních jazyků "procedurálního typu" - popisuje pomocí příkazů. Posloupnost příkazů v podstatě tvoří výkonnou část každého programu. Příkazy mohou operovat na proměnných určených v deklarační části programu nebo na implicitních proměnných určených po skupinách specifikací ARRAY, popř. na systémových proměnných, které se nedeklarují. Příkazy mohou operovat také na výrazech kombinujících proměnné uvedených kategorií a volání funkcí.

#### 3.2.10.1 Přiřazovací příkaz

Exekuční přiřazovací příkaz přiřazuje proměnné nebo všem proměnným seznamu cílových proměnných jejichž identifikátory jsou zapsány vpravo od operátoru ">" hodnotu získanou vyhodnocením výrazu zapsaného vlevo od tohoto operátoru. Zvláštním případem výrazu je proměnná nebo volání funkce.

Pro komunikační přiřazovací příkaz platí zcela obdobná syntaktická pravidla, jen operátor ">" je nahrazen operátorem "<", zamění se slova "vpravo" a "vlevo" a není přípustné přiřazovat zároveň několika proměnným. Vyhodnocení výrazů a vlastní přiřazení se však neprovádí v exekutivě dané sítě pracující v mikropočítačové řídicí stanici v reálném čase, ale probíhá pod řízením programu COLEDA v operátorské stanici a případně s využitím sériové linky, sběrnice nebo prostřednictvím modemů a telefonní sítě.

Zvláštním typem komunikačního přiřazování je vlastně i zobrazování hodnot resp. stavů proměnných pomocí zobrazovací jednotky operátorské stanice.

### Příklady

Exekuční přiřazení:

```
ZadanaPoloha > Vystup04;  
ZadanaPoloha + Posun > Vystup04;  
Povel > !Vystup007; /*Negace vystupu*/  
a * 3.4 > b, bb, bbb, c, d, dd, ddd;  
K & L | !M > B, BB, BBB;
```

Komunikační přiřazení:

```
ZadanáPoloha < 120; Posun < 12 ;
```

Zobrazování hodnot proměnných:

```
ZadanaPoloha > ___; /*na podtržitka*/
```

### Konverze typu proměnných

Při každém přiřazení při kterém typ výrazu neodpovídá typu proměnné, které má být přiřazena hodnota, dochází automaticky ke konverzi typu, nebo hlásí překladač chybu.

Rovněž před provedením aritmetických a logických operací jakož i vyhodnocení relací, probíhá při různém typu operandů konverze. U relací se konvertuje na jednodušší typ (FIXP-FLTP na FIXP-FIXP), u ostatních operací na vyšší typ. Pokud požadujeme zachování přesnosti při vyhodnocení relací, je nutné předem přiřadit hodnotu proměnné uchovávané s menší přesností jiné proměnné, která byla deklarována jako vyšší typ, tzn. provést konverzi explicitně.

Konverze mezi typy FIXP a BYTE probíhají se saturací do rozsahu <0, 255>. Při obráceném směru konverze k saturaci pochopitelně nemůže dojít.

Další informace o kompatibilitě typů proměnných a konverzi jsou uvedeny v rámci obecného popisu standardních funkcí.

### 3.2.10.2 Podmíněný příkaz

Jazyk LEDA zahrnuje příkaz IF zapisovaný takto:

```
IF (Výraz) Příkaz
```

nebo:

```
IF (Výraz) Příkaz1 ELSE Příkaz2
```

Výraz je vyhodnocen jako BOOL, má-li nenulovou hodnotu je následující Příkaz resp. Příkaz1 proveden, v opačném případě příkaz IF končí, resp. je proveden Příkaz2 zapsaný za ELSE.

**Příklady:**

```
IF (RezimAUT & !Blokada & (Hlad12>=4560)) 8000 > ZadanaPoloha;  
IF (MrazOchrana) 0000 > ZadPolKlapky ELSE VystPI > ZadPolKlapky;
```

### 3.2.10.3 Příkazy cyklu

#### Příkaz WHILE

Příkaz WHILE se zapisuje takto:

### WHILE (Výraz) Příkaz

Výraz je vyhodnocen jako BOOL, má-li nenulovou hodnotu, je Příkaz vykonán a řízení je předáno na nové vyhodnocení Výrazu, ... cyklus se opakuje až do okamžiku, kdy je vyhodnocena nepravdivost Výrazu (hodnota 0) - příkaz WHILE je potom ukončen.

#### Příklad

```
WHILE (Pole1[i]<=Pole2[i]) i + 1 > i;
```

### Příkaz FOR

Příkaz FOR se zapisuje takto:

```
FOR (Příkaz1;Výraz;Příkaz2) Příkaz3
```

Příkaz1 je inicializační - může chybět

Výraz je podmínkou pokračování cyklu (jako u WHILE)

Příkaz2 je určen pro reinitializaci - může chybět

Příkaz3 se provádí opakovaně dokud má Výraz nenulovou hodnotu a vždy po jeho vykonání se provede Příkaz2. Příkaz3 může chybět.

#### Příklad

```
FOR (1>i; (Pole1[i]<<Pole2[i]) & (i<=18); i+1>i) Pole1[i]+Sum > Sum;
```

### 3.2.10.4 Příkaz SWI

Příkaz SWI (přepínač) přenáší řízení na jeden z několika následujících příkazů, a to v závislosti na hodnotě výrazu, který je součástí příkazu SWI. Je tedy možné jedním příkazem SWI rozvětvit program na různý počet větví - tento počet je v některých implementacích omezen na 60.

Příkaz SWI se zapisuje takto:

```
SWI (Výraz) Příkaz
```

přítom Příkaz je prakticky vždy složeným příkazem a libovolný příkaz uvnitř tohoto příkazu může být označen speciálním návěštím ve tvaru:

**konstanta:**

V jednom příkazu SWI se nesmí vyskytnout dvě stejné konstanty v návěštích; všechny konstanty musí být stejného typu jako Výraz.

Nejvýše jeden příkaz uvnitř složeného příkazu ve SWI může být označen klíčovým slovem DEFAULT jako návěštím.

Příkaz SWI vyhodnotí výraz v závorce a jeho hodnotu porovnává se všemi konstantami v jednotlivých návěštích. Jestliže dojde k rovnosti, pokračuje program vykonáním daného příkazu, a po jeho ukončení je ukončen celý příkaz SWI.

Jestliže hodnota výrazu není rovna žádné konstantě uvedené v návěštích, předává příkaz SWI řízení na první příkaz za návěštím DEFAULT:. Není-li žádný příkaz označen DEFAULT, je v tomto případě řízení přeneseno za příkaz SWI.

Zvláštním případem použití SWI je binárně řízený přepínač - výraz za SWI může být typu BOOL a následující složený příkaz může mít jen dva příkazy s návěštím "0:" a "1:" (v libovolném pořadí).

Příklad:

```
SWI (Klavesa)
```

```
{ KodCR: 1 > Navrat;
```

```
KodLF: 1 > NovyRadek;  
KodALM: {1 > RezimALM; 0 > Houkej };  
};
```

### 3.2.10.5 Příkaz *BREAK*

Příkaz *BREAK* způsobí ukončení nejnvnitřnějšího příkazu *WHILE* nebo příkazu *FOR* nebo složeného příkazu či bloku.

### 3.2.10.6 Příkaz *CONT*

Příkaz *CONT* způsobí přenesení řízení na konec nejnvnitřnějšího příkazu *WHILE* nebo *FOR*.

### 3.2.10.7 Příkaz *RET*

Příkaz *RET* (return) vrátí řízení z vnitřku funkce do místa odkud byla funkce vyvolána.

### 3.2.10.8 Příkaz *GOTO*

Příkaz nepodmíněného skoku na návěští zapsané za *GOTO*. Překladač povolí pouze příkazy *GOTO* uvnitř sítě, uvnitř těla funkce nebo reprezentace nebo uvnitř některého složeného příkazu v příkazu *WHILE*, *FOR*, *SWI*. Znamená to, že nelze např. skočit zvenčí do těla funkce nebo cyklu a nelze tělo funkce nebo cyklus pomocí *GOTO* opustit.

## 3.2.11 Složený příkaz

Složený příkaz je skupina příkazů oddělených středníky uzavřená mezi příkazové závorky "{" a "}". Všude, kde smí být použit příkaz, smí být použit složený příkaz.

## 3.2.12 Návěští

Každý příkaz může být opatřen předřazeným návěštím. Návěští je identifikátor následovaný znakem ":". Návěští je povinnou součástí deklarace funkce resp. deklarace reprezentace - určuje identifikátor funkce resp. reprezentace. V rozsahu povolených skoků *GOTO* musí být každé návěští jedinečné, je tedy např. přípustné v různých funkcích používat totožná návěští.

Z uvedeného vyplývá, že před libovolný příkaz lze napsat bez použití komentářových závorek "/\*", "\*/" nadbytečné návěští jako komentář prakticky libovolné délky (ukončený dvojtečkou). Připomeňme, že překladač bude dvě návěští považovat za různá, jestliže se liší alespoň v jednom z prvních dvanácti znaků.

## 3.2.13 Standardní funkce

Standardní funkce uživatel nedeclaruje - má ovšem možnost deklarovat lokální nebo i globální funkci s identifikátorem shodným s identifikátorem standardní funkce a tím lokálně nebo i globálně ve svém programu standardní funkci "překrýt".

Standardní funkce mají až na několik výjimek pevně určen počet a typ vstupních a výstupních parametrů - při volání probíhá automaticky konverze proměnných stejně jako u výrazů. Počet parametrů je v některých případech nulový.

Volání funkce (standardní i uživatelské) se skládá postupně:

- ze seznamu vstupních parametrů vzájemně oddělených znaky ",",
- ze znaku ">"
- z identifikátoru funkce

Identifikátor funkce může vystupovat stejně jako identifikátor proměnné ve výrazech, což znamená, že může tvořit také skutečný vstupní parametr jiné standardní nebo deklarované funkce. Nejčastěji tvoří identifikátor funkce výraz na levé straně exekučního přiřazovacího příkazu a je proto následován dalším znakem ">" a dále seznamem výstupních parametrů.

Vstupním parametrem funkce tedy může obecně být:

- skalární parametr
- výraz

Připomeňme, že zvláštním případem výrazu je:

- proměnná
- konstanta
- volání funkce

Výstupním parametrem funkce může být:

- skalární parametr

a tedy také:

- proměnná

Výstupním parametrem může být ve zvláštních případech také výraz, nebo konstanta (výraz určující adresu nebo adresová konstanta).

Počet skutečně zadaných vstupních a výstupních parametrů může být u vybraných standardních funkcí proměnný (NLT, MAX). U ostatních funkcí musí být vždy zachován určený počet oddělovačů ",",

U funkce s jedním výstupním parametrem nemusí být za identifikátorem funkce zadáno ">" ani následující výstupní parametr a identifikátor funkce může přímo vystupovat ve výrazech nebo jako vstupní parametr jiné funkce.

Obecnou zásadou uspořádání seznamů parametrů je umístění případného parametru určujícího mód činnosti funkce - nebo skupiny takových parametrů - na konec seznamu vstupních parametrů, tedy bezprostředně před znak ">". Na stejném místě se uvádí parametr udávající časové zpoždění resp. interval (DSR, SDR, MSE, TDL, MAV, MMA, MMI) nebo konstanta udávající dimenzi pole statických proměnných (SRG, REC). Rovněž parametry časových konstant se zapisují do druhé části seznamu vstupních parametrů až za parametry udávající adresy vstupních signálů (PID, PDC, FIL, PIM), ale před případné parametry výstupních limitů (u funkcí PID, PDC).

Pro každou standardní funkci je pevně stanoveno zda je dynamická nebo statická. Každá realizace dynamické funkce má vyhrazenou staticky určitou oblast paměti pro záznam a uchování stavového vektoru funkce.

### Kompatibilita typů a konverze

Všechny uživatelské a některé standardní funkce mají pevně určen typ vstupních a výstupních parametrů (formálních parametrů). V jednotlivých voláních je přípustné odchýlit se typem skutečných parametrů funkcí od typu daného deklarací nebo vlastnostmi standardní funkce; v některých případech hlásí při nesouladu typů překladač chybu "nekompatibilní typy". Obecně platí tato pravidla:

- Mezi aritmetickými proměnnými (typy BYTE, FIXP, FLTP) probíhají konverze automaticky, s případnou saturací na mezní hodnoty, tj. do intervalu:

<0,255> pro typ BYTE

<-32768,32767> pro typ FIXP

- Proměnné typu BOOL mají k proměnným typu FIXP a FLTP vztah stejný jako číselná konstanta 1 resp. 0 - automatická konverze probíhá v obou směrech.
- Ve výrazech se u dvojice proměnných svázaných relačním operátorem (<=, ==, !=, ...) konvertuje na nižší typ (FLTP na FIXP, ...); pro ostatní binární operace (aritmetické a logické) naopak na vyšší ("delší") typ. Desetinná tečka v číselné konstantě určuje typ FLTP; konstanty zadané jako celočíselné jsou chápány jako FIXP.

Soubor standardních funkcí konkrétní verze ZPV zpravidla zahrnuje:

Identifikátor	Určení funkce
<u>IPR</u>	vstupní zpracování dat přijímaných do stanice
<u>OPR</u>	výstupní zpracování dat odesílaných ze stanice
<u>ATN</u>	převod pole znaků na číselnou hodnotu

<u>OUTS</u>	generace znakového pole číselných údajů
<u>OUTR</u>	předání zprávy k vyslání sériovou linkou
<u>OUTF</u>	výstup řetězců do diskového souboru
<u>WRITEC</u>	zápis záznamu do binárního archivního souboru
<u>READC</u>	načtení skupiny záznamů z binárního archivního souboru
<u>AKTC</u>	aktualizace archivního souboru novými záznamy
<u>LASTTC</u>	načtení časového údaje poslední aktualizace souboru
<u>BEEP</u>	tónový výstup
<u>AUDIO</u>	obecný tónový výstup

Posledních 7 funkcí je použitelných jen v operátorské stanici (osobním počítači).

Knihovna standardních funkcí LEDA využitelných jak v operátorské stanici, tak i v procesní stanici v současné době zahrnuje následující funkce:

Identifikátor	Určení funkce
* <u>PID</u>	regulátor PID resp. PSD (PI,I resp. PS,S)
* <u>PDC</u>	regulátor PD
* <u>PIM</u>	pulzní intervalový modulátor (šířkový a frekvenční)
<u>NLT</u>	statická nelineární transformace (zadaná tabulkou)
<u>SAT</u>	saturace signálu
<u>DBN</u>	pásmo necitlivosti
* <u>FIL</u>	filtr 1.řádu (dolní propust)
* <u>DIF</u>	diference časové řady (pro analogové i dvouhodnotové vstupy)
* <u>SUM</u>	sumace časové řady
<u>SRG</u>	posuvný registr tvořený jednodimenzionálním polem číselných proměnných
* <u>MAV</u>	klouzavý průměr
* <u>MMA</u>	maximum na klouzavém intervalu
* <u>MMI</u>	minimum na klouzavém intervalu
<u>MAX</u>	okamžité maximum
<u>MIN</u>	okamžité minimum
<u>SQT</u>	druhá odmocnina
* <u>TDL</u>	časové (dopravní) zpoždění
<u>ABS</u>	absolutní hodnota
* <u>CMP</u>	třípásmový komparátor s hysterezí
<u>SEL</u>	výběr signálu řízený dvouhod. vstupem - přepínač
<u>SET</u>	nastavení dvouhodnotového výstupu
<u>RES</u>	shození dvouhodnotového výstupu
* <u>DSR</u>	zpožděný náběh dvouhodnotového signálu
* <u>SDR</u>	zpožděné odpadnutí dvouhodnotového signálu
* <u>IMP</u>	impuls z náběžné hrany
* <u>NIMP</u>	impuls ze sestupné hrany
* <u>MSF</u>	monostabilní klopný obvod
* <u>CNT</u>	čítač
* <u>REC</u>	záznam hodnoty proměnné do cyklického zásobníku
* <u>CER</u>	vyvolání hodnot z cyklického zásobníku do pole
<u>CPS</u>	sdružování skupin BOOL proměnných
<u>DCPS</u>	rozklad proměnných na skupiny BOOL proměnných

Funkce označené "\*" jsou dynamické, tzn., že zpracovávají vedle okamžitých hodnot vstupních parametrů ještě další hodnoty závislé na "historii vstupů" nebo na starší hodnotě vlastního výstupu. Každé volání takové funkce vytváří pro interpret objekt s vlastním staticky alokovaným stavovým vektorem (CMP operuje pouze na vlastních výstupech bez zvláštního stavového vektoru). Tyto funkce nelze korektně používat s indexovanými vstupy nebo výstupy v těle cyklu.

Soubor standardních funkcí jazyka zahrnuje také funkce pro převod hodnot skupiny dvouhodnotových proměnných na hodnoty aritmetické proměnné a zpět. Jejich užití je v některých případech zbytečné, protože implicitní proměnné A, O, S, X, Z umožňují kombinovat přístup slovní s přístupem k jednotlivým bitům bez specializovaných prostředků. Lze tedy v LEDA testy na určité kombinace hodnot BOOL proměnných místo BOOLských výrazů vyhodnocovat formou testu na číselnou konstantu (jako relaci ve výrazu nebo v příkazu SWI přímo) a naopak také nastavovat jedním přiřazovacím příkazem celou skupinu až šestnácti dvouhodnotových veličin, resp. osmi dvouhodnotových veličin v poli OL.

Identifikátory standardních funkcí nejsou, jak si lze všimnout, kolizní s jakýmkoli přípustnými identifikátory implicitních nebo systémových proměnných. Pokud chce uživatel těchto standardních funkcí využívat, musí zajistit, aby rovněž jím deklarované lokální proměnné nepoužívaly uvedené identifikátory. Pro jména zavedená jako ekvivalentní označení implicitních proměnných (pomocí "=") musí uživatel zabránit kolizi s identifikátory standardních funkcí bezpodmínečně, neboť prioritou lokálního pokrytí se nemůže u globálních identifikátorů uplatnit.

Lze např. doporučit, aby identifikátory uživatelem deklarovaných proměnných, funkcí i reprezentací nebyly tříznakové - tím odpadne většina obav z možné kolize.

### 3.2.14 Uživatelské funkce

Uživatel může deklarovat vlastní funkce, a to jako:

- globální
- lokální v bloku

Tělo sítě i funkce je tvořeno blokem, který může mít deklarované vlastní funkce. Funkce lokální v bloku proto mohou být zároveň:

- lokální ve funkci

Funkce globální je nutné deklarovat za záhlavím sítě, před blokem tvořícím tělo sítě, funkce lokální v bloku se deklarují na začátku bloku za první příkazovou závorkou "{" za deklaracemi lokálních proměnných a deklaracemi lokálních reprezentací.

Deklarace funkce je uvozena:

- návěštím tvořeným identifikátorem funkce a dvojtečkou;

dále následuje:

- seznam formálních parametrů (vstupů funkce) bezprostředně následovaný znakem ">" - nepovinně;
- klíčové slovo FUN;
- znak ">" bezprostředně následovaný seznamem formálních parametrů (výstupů funkce) -nepovinně;
- deklarace typu formálních parametrů;
- blok tvořící tělo funkce;
- znak ";" ukončující deklaraci funkce.

Seznam formálních parametrů je řada identifikátorů proměnných resp. polí oddělených vzájemně znaky ",", ". Překladač určuje pro každé volání funkce zvlášť, zda příslušný skutečný parametr je adresou, ze které má být přečtena hodnota ukládaná do zásobníku pro zpracování v těle funkce, nebo zda jde o konstantu, která má být do zásobníku uložena přímo jako hodnota skutečného parametru.

Z uvedeného vyplývá, že se v jazyce LEDA pracuje mechanismem "volání parametrů hodnotou", což znamená, že pro potřeby těla funkce se do zásobníku vkládají při vyvolání funkce kopie hodnot - přímo nebo nepřímo určených všemi skutečnými parametry - a při návratu z funkce (RET) se vrátí - jen do proměnných určených výstupními parametry - hodnoty modifikované v těle funkce. Znovu zdůrazněme, že při vyvolání funkce se do zásobníku zapíše jak vstupní, tak i výstupní hodnoty a tím se určí mj. počáteční stav výstupů.

V bloku tvořícím tělo funkce mohou být deklarované některé lokální proměnné jako statické (klíčové slovo STAT), funkce má potom možnost realizovat dynamické chování bez závislosti na globálně dostupných proměnných. Dynamické chování mohou ovšem mít i funkce, jejichž výstupy jsou zároveň stavovými veličinami, a to i bez lokálních statických proměnných. Proměnné, které nejsou statické lze používat v těle funkce jen za předpokladu, že byly deklarované jako v dané funkci lokální!

V těle deklarovaných funkcí je povoleno používání všech standardních funkcí jazyka LEDA včetně funkcí dynamických. Pro každé volání deklarované funkce je překladačem vyhrazena statická oblast paměti v rozsahu daném součtem délek stavových vektorů všech volaných standardních funkcí. Při přípravě volání deklarované funkce jsou všechna "historická data" přenesena do jediné oblasti adresované všemi dynamickými funkcemi volanými v jejím těle; po návratu z funkce jsou modifikované stavové vektory přeneseny zpět do uvedené statické oblasti paměti.

Typy proměnných tvořících parametry funkce určují deklarace typu bezprostředně předcházející bloku tvořícímu tělo funkce. Deklarace jsou tvořeny seznamem klíčových slov typů proměnných oddělených čárkami a odpovídajících jednotlivým parametrům seznamu vstupních parametrů a (bez zvláštního oddělování) parametrům seznamu výstupních parametrů. V těchto deklaracích se klíčové slovo **STAT** nesmí vyskytovat.

Volání uživatelské funkce se řídí stejnými pravidly jako volání standardních funkcí popsané v 3.2.13..

### Příklady:

Deklarace velmi jednoduché funkce zajišťující linearizaci signálu z odporového teploměru s převodem na FIXP údaj v desetínách 'C:

```
Lin2: Vstup > FUN > Vystup;          /* Linearizace pro Pt100 do 200'C */
      FIXP,          FIXP
      { Vstup,
        0000,  -0500,
        1653,   0000,
        3277,   0500,
        4875,   1000,
        6450,   1500,
        8000,   2000
        > NLT > Vystup;
      };
```

Vyvolání této funkce včetně zobrazení hodnot vstupu a výstupu (místo proměnných jsou použity skalární parametry):

```
_____ < AI07 > Lin2 > TTV1 > TT$ _____;    /* Tepl.topne vody1 */
_____ < AI08 > Lin2 > TTV2 > TT$ _____;    /* Tepl.topne vody2 */
```

Deklarace funkce se třemi vstupy a dvěma výstupy - funkce zajišťuje blokování rozporných povelů operátora a vylučuje okamžitou reverzaci servomotoru:

```
Tpo: Man, PovelMene, PovelVice > FUN > VystupMene, VystupVice;
      BOOL,      BOOL,      BOOL,      BOOL,      BOOL
      { BOOL StaraVice;
        IF (Man) { VystupVice > StaraVice;
                   PovelVice & !VystupMene > VystupVice;
                   PovelMene & !StaraVice & !VystupVice > VystupMene;
                 };
      };          /* Konec deklarace funkce Tpo */
```

Vyvolání funkce Tpo:

```
ManV1, MeneV1, ViceV1 > Tpo > V1Mene, V1Vice;
```

Nebo se zobrazením hodnot všech vstupních a výstupních proměnných:

```
_ < ManV1, MeneV1 > _, ViceV1 > _ > Tpo > V1Mene > _, V1Vice > _;
```

### Oblast viditelnosti funkcí

Každá funkce je v programu použitelná počínaje blokem její vlastní deklarace s případným dalším omezením na blok, tj. také síť nebo podprogram, ve kterém je funkce lokální. Pro funkce a reprezentace standardní obdobné omezení neexistuje - jsou globální v celém programu.

### 3.2.15 Bloky

Blok je složený příkaz, který je doplněn za příkazovou závorkou "{" alespoň jednou deklarácí proměnné nebo deklarácí reprezentace nebo deklarácí funkce (v tomto pořadí, které umožňuje deklarované reprezentace využívat v deklarácích funkcí téže sítě).

### 3.2.16 Sítě

Sít' je v podstatě blok (nebo skupina bloků) doplněný nepovinně deklarácemi proměnných, reprezentací a funkcí a dále doplněný údaji soustředěnými v záhlaví sítě. Sítě jsou spouštěny cyklicky s periodou, uvedenou v záhlaví sítě. Existují dvě kategorie sítí:

- Prioritní sít' programu je vždy právě jedna. **Tato sít' je v programu uvedena jako první a musí mít periodu shodnou se základní periodou** zpracování dat zadávanou v záhlaví programu (**PERIOD**). Význačnou vlastností této sítě je, že v případě potřeby může přerušovat běh ostatních sítí. Slouží pro zpracování krátkých obslužných programů např. pro vstupní a výstupní stranu, ovládání displeje atd.
- Cyklické sítě jsou všechny ostatní sítě programu. Jsou spouštěny v pořadí v jakém jsou napsány v programu v závislosti na periodách sítí.

Záhlaví každé sítě je tvořeno:

- identifikátorem sítě
- seznamem parametrů sítě uzavřených mezi kulaté závorky:
- konstantou udávající periodu spouštění cyklické sítě - nezadání tohoto parametru určuje sít' acyklickou
- konstantou udávající fázi spouštění cyklické sítě - pro acyklickou sít' se nezadává
- skalárním parametrem typu BOOL s významem Enable - uvolnění sítě do exekutivy stanice; nezadaný parametr má význam jako Enable == 1

Čtvrtý parametr sítě je v současných implemetacích prázdný - jde o rezervu pro stavový vektor sítě. Je tedy za třetím parametrem sítě nutné zapsat čárku a za ni uzavírací závorku ).

Za záhlavím sítě následuje znak ";", dále případné deklarace globálních funkcí (volatelných z jiných sítí) a blok tvořící tělo sítě.

Celá deklarace sítě je ohraničena klíčovými slovy **BEGIN** a **END**.

Jestliže následuje za klíčovým slovem **BEGIN** klíčové slovo **MASTER**, je celá tato sít' vykonávána v exekutivě operátorské stanice (PC). Spouštění může být periodické - není však zabezpečena systémovými prostředky synchronizace s exekutivou "procesní" stanice.

Všechny cyklické sítě procesní stanice mohou být spouštěny jen v periodě, která je celočíselným násobkem základní periody zpracování cyklických sítí (zadává se v záhlaví programu). Fáze spouštění sítě je časový údaj, který vydělen periodou obslužné sítě lokální sběrnice udává počáteční hodnotu čítače základních period pro spouštění dané sítě. Tento čítač je na hodnotu určenou fází sítě nastaven jen po restartu stanice; v běžném provozu je čítač nastavován na hodnotu určenou periodou sítě vždy po dosažení nulové hodnoty čítače po jeho dekrementování při každém spuštění obslužné sítě lokální sběrnice. Zároveň je dosažení nulové hodnoty čítače nutnou podmínkou spuštění sítě; další podmínkou je stav Enable == 1 a ukončení zpracování všech těch cyklických sítí se splněnými podmínkami spuštění, které byly v programu deklarovány výše, než uvažovaná sít'. Další informace o spouštění sítí jsou uvedeny v kapitole 8.2.

### 3.2.17 Ekvivalence řetězců

Před první sítí programu lze zapsat tzv. ekvivalence řetězců - přesněji ekvivalence řetězcových konstant, které určují jakým řetězcem má být nahrazen preprocesorem překladače LEDA každý výskyt určitého řetězce znaků zdrojového textu vyhovujícího syntaktickým požadavkům na identifikátor (nahrazovaný řetězec musí mj. začínat písmenem). Z nahrazování řetězců jsou vyloučeny všechny řetězcové konstanty uvedené v programu počínaje **BEGIN** první sítí, kde prohledávání a případné náhrady začínají.

Ekvivalence se zapisuje takto:

*řetězc.konst. nahrazujícího řetězce == řetězc.konst.nahrazovaného řetězce*

Obvyklým použitím ekvivalencí řetězců je přiřazení uživatelských identifikátorů implicitním proměnným. Ekvivalence dávají možnost zavádět také symbolické konstanty a konstantní makrosymboly. V oblasti zápisu ekvivalencí řetězců není nutné uzavírat komentáře mezi dvojnaky /\* a \*/. Lze tedy například zadat tyto ekvivalence:

```
"F1C" == "Zesileni3"    /* Zesilení 3.regulatoru PID */
"6"    == "PocetObv"   /* Pocet regulacnich obvodu */
"20"   == "BROWN"     /* Kod barvy VGA pro reprezentace */
```

### 3.2.18 Program

Program je tvořen množinou sítí, které předchází záhlaví programu a nepovinně množina ekvivalencí řetězců.

Program - resp. jeho záhlaví - začíná klíčovým slovem PROGRAM, následuje libovolný text (komentář) a dále nepovinně klíčové slovo ARRAY, za kterým následuje specifikace implicitních polí, tj. seznam písmen určujících dostupná pole implicitních proměnných. Jednotlivá písmena jsou oddělena čárkami; alespoň jedno písmeno musí být uvedeno. Seznam identifikátorů polí implicitních proměnných je ukončen znakem ";".

Pro programy, které jsou součástí tzv. projektu distribuovaného řídicího systému, se v záhlaví programu zadává adresa dané stanice na globální sběrnici - toto číslo je zároveň logickým číslem stanice používaným při komunikaci po telefonní síti. Zadává se jako dvouciferné hexadecimální číslo zapsané za klíčové slovo BUS a následované znakem ";". Číslo BUS ovlivňuje postavení stanice v projektu jako stanice MASTER (číslo BUS <= 0F) nebo stanice SLAVE (podrobněji ve zvláštní kapitole v rámci popisu implementace pro konkrétní řídicí systémy).

Pro stanice připojené prostřednictvím modemů k telefonní síti se zadává za klíčové slovo TEL řetězcová konstanta telefonního čísla včetně případného čísla telefonního uzlu pro meziměstské spojení (podrobnosti jsou uvedeny v rámci popisu standardních funkcí PUTMSG a GETMSG zajišťujících řízení komunikace přes telefonní síť).

Nepovinným parametrem, který slouží pro řízení překladu, je řetězcová konstanta zadávaná za klíčové slovo UNIT a následovaná opět znakem ";". Řetězec UNIT udává typ řídicí stanice, pro kterou je program určen a potlačuje vliv volby typu stanice zadávané v "Options-Compiler" vývojového prostředí COLEDA (zadávané např. "CCSA", "T2008C", "T2008M", "DAPU10"). Zadání parametru UNIT v záhlaví programu je významné v projektech zahrnujících různé typy řídicích stanic různých systémů, na které byl jazyk LEDA implementován.

Povinným parametrem celého programu je základní perioda zpracování cyklických sítí udávaná jako desetinné číslo v sekundách za klíčovým slovem PERIOD. Za řetězec konstanty lze zapsat symbol fyzikální jednotky "s" a zápis se opět ukončuje znakem ";".

Nepovinně mohou následovat ekvivalence řetězců a dále jedna nebo více sítí. Koncovým znakem programu je klíčové slovo END.

Komentář zapsaný za PROGRAM se - na rozdíl od všech ostatních komentářů vyskytujících se ve zdrojovém textu - přenáší do produktu překladu (do jinak binárního datového souboru s příponou BED nebo PED) jako jeho prvních 64 byte a může sloužit pro identifikaci datových souborů, které mohou být (BED) přeneseny do paměti procesní stanice. Tento komentář by proto měl zahrnovat jednoznačné označení verze programu.

### 3.2.19 Direktiva INCLUDE

Preprocesor překladače LEDA zpracovává direktivu:

```
#INCLUDE "Jméno_souboru"
```

která způsobí před překladem načtení určeného textového souboru do překládaného programu na místo direktivy INCLUDE.

Vkládaný soubor, který určuje Jméno\_souboru (včetně označení disku a posloupnosti adresářů), může obsahovat libovolný úsek zdrojového textu psaného v jazyce LEDA, například ekvivalence řetězců, deklarace proměnných nebo funkcí.

V libovolných místech programu smí být použit i větší počet direktiv INCLUDE odkazujících na různé nebo i stejné vkládané soubory.

Používání direktiv INCLUDE zkracuje text programu zobrazovaný po spuštění interpretace ve vývojovém prostředí COLEDA resp. pomocí programu TOLEDA a umožňuje tak zvýšit rezervu kapacity operační paměti osobního počítače. Výhodné může být rovněž "ukrytí" zdrojového textu pro operátora.

### 3.3 Práce se znakovými proměnnými, poli a řetězci

Implicitní proměnné typu `CHAR` neexistují, a proto musí být všechny znakové proměnné, jak pro užití v sítích podřízené řídicí stanice, tak i v MASTER sítích, explicitně deklarovány jako automatické proměnné (dočasného významu), např.:

```
CHAR Identifikátor1, Identifikátor2;
```

nebo jako statické proměnné (s trvale vyhrazenou paměťovou buňkou):

```
STAT CHAR Identifikátor3, Identifikátor4;
```

Vedle znakových polí (i,j jsou číselné konstanty BYTE nebo FIXP):

```
CHAR ARRAY Ident5[i..j];          /* Každé pole zvlášť */
```

a statických znakových polí:

```
STAT CHAR ARRAY Ident6[i..j];
```

mohou být deklarována také statická znaková pole ležící v paměťové oblasti některého implicitního nebo dříve deklarovaného pole, např.:

```
STAT CHAR ARRAY PoleCh[0..31] EQU IR100;
```

(zde se jedná o ekvivalenci adres s částí implicitního pole proměnných typu BYTE od IR100 do IR11F)

nebo:

```
STAT CHAR ARRAY PoleCh[0..31] EQU P40;
```

(zde se jedná o ekvivalenci s 16ti prvky pole P typu FIXP).

Statické proměnné typu CHAR alokované v prostoru některého inicializovaného pole implicitních proměnných ( S, P, W, F) a všechny statické proměnné MASTER sítí jsou při studeném startu nastaveny na počáteční hodnotu 0 nebo na hodnotu určenou komunikačním přiřazením:

```
Identifikátor < "Znak";
```

nebo: 

```
PoleCh[Konst] < "Znak";
```

přičemž na pravé straně je znaková konstanta, tj. vlastně řetězcová konstanta s délkou 1, např.:

```
"A" | "B" | "Z" | "a" | "!" | "+" | " "
```

nebo jiné tisknutelné znaky ASCII souboru, popř. i netisknutelné znaky, které mohou být zapsány ve tvaru:

```
"#XY"
```

kde X a Y jsou hexadecimální číslice tvořící ASCII kód znaku, nebo často používané netisknutelné znaky přechodu na nový řádek jako:

```
"/N"      /* line feed */
```

a přechodu na novou stránku jako:

```
"/F"      /* (form feed) */
```

Pomocí "#XY" lze inicializovat znakovou proměnnou i hodnotami (8bitovými), které nepatří do ASCII souboru.

Proměnným typu CHAR lze přiřazovat znakové konstanty také exekučně, např. přiřazovací příkaz:

```
"2" > PoleCh[6];
```

zapíše do šestého prvku pole PoleCh ASCII kód číslice 2, tj. číslo 50 dekadicky, resp. 32 hexadecimálně.

Hodnoty znakových proměnných lze přiřazovat jiným proměnným:

```
Identifikátor > PoleCh[i];
```

a lze je komparovat, např.:

```
IF (PoleCh[i] == PoleCh[j]) ...
```

resp.:

```
IF (PoleCh[i] != PoleCh[j]) ...
```

Jiné operace nejsou na proměnných typu CHAR povoleny.

Vedle znakových proměnných lze deklarovat v jazyce LEDA také řetězcové proměnné umožňující snazší provádění některých manipulací se skupinami znaků a operace na nich.

Vnitřní reprezentací řetězcové proměnné je pro jazyk LEDA skupina byte, z nichž první obsahuje údaj délky řetězce N v rozsahu 0 až 255 a dalších N byte zachycuje vlastní znaky řetězce. Řetězec může být obecně naplněn libovolnými 8bitovými údaji, které nemusí patřit do ASCII souboru. Koncový znak se u řetězců v LEDA nepoužívá, některé byte mohou proto být i nulové aniž by došlo ke zkrácení řetězce.

Řetězcové proměnné musí být deklarovány, např. takto:

```
STAT CHAR#8 Retez1, Retez2;
```

Tato deklarace vyvolá alokaci statické oblasti paměti RAM v celkové délce 18byte pro 2 řetězce maximální délky 8 znaků.

Jednotlivé znaky řetězcové proměnné i její délku lze modifikovat prostřednictvím adresově ekvivalentního znakového pole nebo pole typu BYTE zavedeného např. deklarací:

```
STAT CHAR ARRAY PoleCH[0..8] EQU Retez1;
```

Řetězcové proměnné lze inicializovat řetězcovými konstantami např. takto (komunikačním přiřazením):

```
Retez1 < "R.konst";
```

Tímto přiřazením nabude řetězcová proměnná Retez1 délky 7, tedy o 1 menší než je uvedeno v deklaraci. Poslední byte prostoru vyhrazeného pro řetězec zůstane nezměněn. Samotná deklarace řetězcové proměnné nezajišťuje nastavení nultého byte délky řetězce.

Řetězcovou konstantu lze řetězcové proměnné přiřadit rovněž exekučně např.:

```
"JINAK" > Retez1;
```

(zde se obsadí jen 6 byte v paměťovém prostoru řetězce, délka řetězce je 5).

Hodnotu řetězcové proměnné lze přiřadit (exekučně) jiné řetězcové proměnné:

```
Retez1 > Retez2;
```

Řetězce lze dále přiřazovat do znakových polí a znaková pole nebo jejich části lze přiřazovat řetězcům. V obou těchto případech probíhá vedle kopírování skupiny znaků také konverze typu spočívající ve vynechání nebo vytvoření jednoho byte délky řetězce. Řetězce lze podobně jako znakové proměnné porovnávat na rovnost či nerovnost.

Jedinou operací (binární), která je na řetězcových proměnných (a konstantách) povolena, je operace sjednocení řetězců, zapisovaná stejně jako součet proměnných jiných typů, např.:

```
Retez1 + "Vsuvka" + Retez2 > Retez3;
```

V tomto příkladu se za řetězec Retez1 "připojí" (bezprostředně) 6 znaků řetězcové konstanty "Vsuvka" a dále obsah řetězcové proměnné Retez2. Výsledný řetězec je uložen do proměnné Retez3, a to tak jak vznikl popsáním sjednocením za předpokladu, že jeho celková délka je  $\leq$  délce řetězcové proměnné Retez3 uvedené v její deklaraci, příp. je výsledný řetězec zkrácen na maximální délku proměnné Retez3.

Řetězce mohou stejně jako proměnné jiných typů vytvářet pole - řetězcové pole může být deklarováno např. takto:

```
STAT CHAR#15 ARRAY PoleR[0..66];
```

zde se jedná o pole 66ti řetězců max.délky 15ti znaků, tj. o vyhrazení paměťového prostoru v rozsahu  $66 \times 16 = 1056$  byte celkem.

Mezi znakovými proměnnými či řetězci a proměnnými ostatních typů (BYTE, FIXP, BOOL) neprobíhají "automatické" konverze tak jako mezi číselnými proměnnými a proměnnými BOOL navzájem. Nelze tedy přiřazovat

např. proměnné typu BYTE proměnným typu CHAR, obráceně a podobně FIXP do CHAR, ... Číselné proměnné lze převádět na série znaků např. pomocí standardních funkcí [OUTS](#) a [OUTF](#), obrácený převod pomocí funkce [ATN](#).

### 3.4 Podobnost a rozdíly LEDA a jazyka C

Již v úvodní části tohoto materiálu bylo konstatováno, že jazyk LEDA lze jen stěží z hlediska flexibility srovnávat s jazykem C. Řada příkazů LEDA je sice ekvivalentní příkazům C - v LEDA chybí mezi příkazy cyklu jen příkaz "do...while" a příkaz SWI je proti Switch syntakticky jednodušší a nenutí k častému používání příkazu BREAK, ale C nabízí daleko bohatší sortiment operátorů, větší výběr typů proměnných, možnost deklarace vlastních typů včetně strukturovaných a především prostředky pro práci s ukazateli.

Na první pohled se odlišuje program v LEDA od programu v C obráceným směrem přiřazovacích příkazů s operátorem přiřazení ">" místo "=". Podstatné však je, že toto tzv. "exekeční přiřazení" není v LEDA použitelné ve výrazech tak jako v C, což je jednou z příčin pro které nelze psát v LEDA programy v tak zhuštěné (a občas i nečitelné) formě jako v C. Navíc ovšem LEDA povoluje přiřazovat hodnotu výrazu (tj. také proměnné či konstanty) většímu počtu proměnných v jediném přiřazovacím příkazu (na pravé straně je seznam proměnných).

Aritmetické operátory "+", "-", "\*", "/" využívá LEDA shodně s C. Operátory "++", "--", "+=", "-=", ... nejsou v LEDA zavedeny. Operátory bitových operací z C "&" resp. "|" mají v LEDA význam "AND" resp. "OR" pro dvouhodnotové proměnné deklarované s typem BOOL. Operátory bitových posunů v LEDA neexistují. Operátor "&&" není v LEDA zaveden, operátor "||" má v LEDA význam "exclusive OR". Symbolem unární operace negace je v obou jazycích "!". Unární dereferenční operátor "\*" z C v LEDA neexistuje, referenční operátor "&" rovněž ne.

Relace se v obou jazycích zapisují téměř stejně, jen pro ostré nerovnosti se v LEDA používají symboly ">>" resp. "<<" místo ">" resp. "<" v C - důvodem je použití těchto nezdvoujených znaků v LEDA pro exekeční resp. komunikační přiřazení, která jsou daleko frekventovanější.

K problematice typů proměnných je třeba připomenout, že LEDA rozeznává na rozdíl od C samostatný typ Boolské proměnné.

Příkazové závorky "{" , "}" mají téměř stejný význam - bloková struktura v LEDA je obecnější tím, že připouští deklarace funkcí lokálních v bloku a lokálních ve funkci, dává tedy LEDA v tomto směru bohatší možnosti strukturování programu a poskytuje širší možnosti psaní programů s vyšší mírou zabezpečení. Předčasné ukončení bloku resp. složeného příkazu pomocí BREAK je v LEDA možné univerzálně, zatímco v C je omezeno jen na příkazy cyklu a příkaz Switch.

Nezanedbatelným formálním detailem je způsob používání znaku ";" pro oddělování jednotlivých příkazů tvořících složený příkaz. V C se ";" zapisuje i před uzavírací příkazovou závorkou "}", v LEDA se na tomto místě použít nemusí (obdobně jako se nepíše v Pascalu bezprostředně před END). Naopak mezi příkazy se v LEDA píše znak ";" vždy, tedy i v případě, kdy je příkaz příkazem složeným, to znamená, že se středník píše i bezprostředně za složenou závorkou ukončující složený příkaz (...);...) v situaci, kdy je tento příkaz následován dalším příkazem bloku. Nadbytečné středníky za příkazy jsou při překladu interpretovány jako prázdné příkazy.

U deklarací a volání funkcí můžeme pozorovat další významnější rozdíly. Především v LEDA existují vedle funkcí také "reprezentace", které jsou vlastně také funkcemi, ale vykonávanými výhradně v operátorské stanici, a to v závislosti na momentálně zobrazované části textu programu.

V deklaraci funkce se v LEDA používá klíčové slovo "[FUN](#)", v deklaraci reprezentace klíčové slovo "[REP](#)". V obou případech tato klíčová slova oddělují seznam vstupních formálních parametrů (vlevo) od seznamu výstupních formálních parametrů (vpravo). Zabezpečení jazyka přispívá kontrola, kterou překladač označí jako chybné všechny přiřazovací příkazy těla funkce, na jejichž pravé straně figuruje formální parametr uvedený v seznamu vstupních parametrů. Proměnné z výstupního seznamu parametrů mohou být pochopitelně v těle funkce přepisovány, ale smí být také čteny; jsou-li deklarovány jako statické (příznak STAT jako static u C), nebo jde o implicitní proměnné, které jsou globální a statické, mohou tvořit stavové proměnné dynamického systému realizovaného funkcí. Oproti C nemá funkce v LEDA svůj typ - výstupů může být více s různými typy. Příkaz RET (return) nemůže být tak jako v C doplněn o výraz určující hodnotu vracenou funkcí - výstupy musí být zadány explicitně přiřazovacími příkazy.

Ve volání funkce se tak, jak je obvyklé, nahradí formální parametry skutečnými parametry a navíc se klíčové slovo FUN resp. REP nahradí identifikátorem funkce resp. reprezentace - tento identifikátor uvozuje jako návěští deklaraci funkce.

Na rozdíl od C se v LEDA všechny vstupní parametry tvořené identifikátory proměnných - ať již jednoduchých nebo polí - používají k určení vstupních hodnot, které jsou jimi bezprostředně adresovány; vstupující konstanty tvoří vstupní hodnoty příkazů těla funkce přímo. LEDA nezná adresové konstanty a neumožňuje uživateli pracovat s pointerem na datové objekty. Syntax jazyka umožňuje rozlišit konstanty od proměnných (reprezentujících adresu objektu) a překladač zajistí automaticky přístup těla funkce k hodnotě stanoveného typu - jde o volání hodnotou s automatickou dereferencí skutečného parametru funkce; skutečnost, zda se dereference provede (pro proměnnou) či ne (pro konstantní parametr) je vlastností každého volání funkce, nikoli vlastností funkce samé (její deklarace resp. definice).

Pro programátora je způsob práce s parametry funkcí v LEDA ve srovnání s C omezující co do flexibility, ale naproti tomu podstatně jednodušší a zejména bezpečnější.

Možnost vzájemných záměn identifikátorů proměnných a konstant na místě skutečných parametrů funkcí (standardních i uživatelem deklarovaných) je významnou výhodou u parametrů, které musí být seřizovány a pro trvalý provoz postačí konstanty (v paměti ROM), ale v některých případech volání funkce můžeme požadovat trvalou seřizovatelnost operátorem nebo adaptací parametrů.

Volání hodnotou zajišťuje v LEDA konzistenci části obrazu prostředí zpracovávaného funkcí. To znamená, že při opakovaném použití těžce proměnné (tvořící parametr uživatelské funkce) v těle funkce, nemůže dojít působením procesu s vyšší prioritou ke změně hodnoty této proměnné, neboť tělo funkce operuje na kopiích hodnot proměnných tvořících její skutečné vstupní parametry v zásobníku. Rovněž u výstupních parametrů funkcí se v LEDA pracuje s počátečními hodnotami tvořenými kopiemi momentálních hodnot příslušných proměnných v okamžiku volání funkce a při postupné modifikaci hodnot výstupních proměnných v zásobníku proto nehrozí nebezpečí jejich předčasného předání k dalšímu zpracování procesem s vyšší prioritou resp. předčasného výstupu ze stanice (významné při iteračních výpočtech).

Řetězení funkcí v LEDA odpovídá lépe sériovému řazení funkčních bloků ve směru toku signálu, než vnořené volání funkcí v C a ostatních univerzálních jazycích.

Například:

V jazyce C:  $z = \text{Fun3}(\text{Fun2}(\text{Fun1}(a,b,p1,p2),c,p3,p4),p5);$

V LEDA:  $a,b,p1,p2 > \text{Fun1},c,p3,p4 > \text{Fun2},p5 > \text{Fun3} > z;$

Tato vlastnost usnadňuje uživateli přechody mezi signálovým diagramem resp. blokovým schématem obvyklým v projektech MaR a zápisem v jazyce LEDA, který je podle názoru autora pro zamýšlenou oblast užití jazyka čitelnější, neboť např. není třeba čtenáře programu odpočítávat závorky, chce-li určit ke které funkci patří určitý parametr. Rovněž strojový překlad blokového schématu do zdrojového textu v LEDA se díky této vlastnosti poněkud zjednodušuje.

Definování konstant v C pomocí direktivy #define nebo konstrukce využívající klíčové slovo "const" je v LEDA nahrazeno mechanismem tzv. ekvivalencí řetězců. Ekvivalence jsou společné pro celý program a jejich hlavním určením je zavedení alternativních mnemotechnických identifikátorů pro implicitní proměnné a symbolické konstanty; možnost makroprogramování je spíše vedlejším důsledkem zavedení ekvivalencí řetězců.

Inicializace proměnných zavedená v C jako součást deklarace je v LEDA rovněž povolena. LEDA však navíc umožňuje zadávat nejen počáteční hodnoty, ale i hodnoty určované operátorem za běhu programu v libovolném místě textu programu pomocí tzv. skalárních nebo vektorových parametrů. Tyto syntaktické prvky jazyka LEDA mohou nahrazovat identifikátory proměnných ve výrazech a skutečných parametrech funkcí, nebo mohou vystupovat v programu izolovaně. Jsou tvořeny identifikátorem proměnné, identifikátorem reprezentace, která převádí zadávané konstanty do požadovaného tvaru odpovídajícího typu proměnné a symboly určujícími místo pro zadání řetězce konstanty (skupina podtržitek nebo tzv. odkaz na marker). Zároveň mohou skalární nebo vektorové parametry zajistit zobrazování momentálních hodnot proměnných do textu spuštěného programu, a to opět prostřednictvím reprezentací, které převádějí tyto hodnoty do formy řetězce znaků nebo do grafické formy. Jsou tedy přímo součástí jazyka LEDA prostředky, které obvykle nabízí až programové prostředí vývojového systému. Zpracování konstant přiřazených (komunikačním přiřazením) proměnné a alternativně zobrazování momentálních hodnot těžce proměnné lze pod vývojovým systémem COLEDA řídit za běhu programu změnou směru přiřazení, tj. pomocí symbolů "<" a ">" směřujících k proměnné nebo alternativně od ní.

Na rozdíl od C zahrnuje jazyk LEDA přímo celou řadu standardních funkcí specializovaných zejména na problematiku statického i dynamického zpracování analogových a dvouhodnotových signálů. Překladač z LEDA musí totiž mít k dispozici všechny informace o standardních reprezentacích pro jednotlivé parametry těchto funkcí i pro jejich skupiny zpracovávané vcelku; potřebné dosti obsáhlé informace je nepraktické zahrnovat do zdrojových textů. Jako nevýhoda tohoto přístupu se na první pohled jeví obtížnější rozšiřování sortimentu funkcí, zvolená strategie však

napomáhá udržení jednotného stylu knihovny standardních funkcí. Poněkud větší uzavřenost jazyka potlačuje také problémy s rozšiřováním seznamu chybových hlášení překladače, doplňováním funkce Help vývojového systému a případným doplňováním knihovny grafických symbolů pro kreslení blokových schémat. Dále je potřeba si uvědomit, že soubor standardních funkcí není žádoucí rozšiřovat o složitější funkce s delší - popřípadě silně neurčitou - dobou zpracování, neboť standardní funkce není na úrovni uživatelského programu přerušitelná (je však pochopitelně přerušitelná drivery periferních zařízení, které mohou zajišťovat i poměrně komplexní činnosti).

Pro některé standardní funkce jazyka LEDA vytváří překladač pro každé volání unikátní objekt - každý objekt má vedle procedurální části (společné s ostatními objekty shodného typu) vlastní, uživateli přímo nepřístupnou, datovou část s pevně alokovanou paměťovou oblastí pro "stavový vektor" dynamického systému, kterým daný objekt je (integrátory, filtry, čítače, klopné obvody se zpožděním, regulátory s astatismem).

Uživatelé deklarované funkce mohou mít rovněž vlastní "historickou databázi", jejíž strukturu deklaruje uživatel explicitně jako skupinu "přídavných" výstupních parametrů funkce. Pro standardní funkce typu "objekt" volané v těle deklarovaných funkcí, zajišťuje vyhrazení příslušné statické oblasti paměti automaticky překladač.

Program v LEDA má odlišnou strukturu od programu psaného v C a připomíná více klasickou strukturu Pascalu. Identifikátory proměnných a funkcí musí být deklarovány dříve (deklarace splývá s definicí; neexistují prototypy volání funkcí), než je jich použito. Program v LEDA se skládá z tzv. sítí tvořených bloky příkazů; sítě se chovají do jisté míry jako samostatné programy se společnou paměťovou oblastí (implicitních proměnných a deklarovaných globálních proměnných programu) spouštěné pod operačním systémem reálného času. Záhlaví každé sítě soustřeďuje parametry pro řízení (podmíněné periodické) spouštění sítě. V LEDA neexistuje obdoba funkce main jako hlavního programu a neexistují prototypy funkcí.

## 4. Popis standardních funkcí

V této kapitole budou podrobně popsány všechny standardní funkce jazyka LEDA, které jsou v současnosti k dispozici pro systémy TRONIC 2016, TRONIC 2008 a pro kompaktní řídicí stanice řady CCSA. Algoritmy zpracování dat v jednotlivých funkcích jsou popisovány jednak slovní formou, jednak v některých případech zápisem v jazyce LEDA, a to i s využíváním jiných standardních funkcí jazyka a pochopitelně i příkazů jazyka popsaných v předcházejícím textu. Tyto programové fragmenty nezahrnují deklarace proměnných ani ostatní povinné náležitosti programu v jazyce LEDA, jsou proti skutečné realizaci funkcí zjednodušeny, a nelze je proto považovat za ekvivalent příslušných funkcí interpretru XEDA.

Programátorský „styl“ těchto fragmentů není podle názoru autora optimální z hlediska možností jazyka LEDA, ale je volen takovým způsobem, aby i bez podrobnější znalosti jazyka LEDA byly tyto zdrojové texty dobře čtenáři srozumitelné.

### 4.1 MAX, MIN - výběr okamžitého maxima, resp. minima

#### Určení funkce a její charakteristika

Funkce MAX resp. MIN určuje okamžité maximum resp. minimum ze skupiny hodnot vstupních proměnných (popř. i konstant) - výstupní signál tvoří "horní obálku" resp. "dolní obálku" všech vstupních signálů. Počet vstupních parametrů funkce MAX resp. MIN je proměnný.

#### Formáty volání funkcí MAX a MIN

$U_1, U_2, \dots, U_n > \text{MAX} > y$

resp.

$U_1, U_2, \dots, U_n > \text{MIN} > y$

#### Vstupní a výstupní parametry

Typ první vstupní proměnné  $U_1$  určuje typ zpracování dat ve funkci - všechny vstupní signály jsou konvertovány na tento typ; výsledek výběru maxima resp. minima je na závěr konvertován na typ výstupní proměnné  $y$  (včetně případné saturace FIXP nebo BYTE výst.).

### 4.2 CMP - komparátor

#### Určení funkce a její charakteristika

Funkce CMP porovnává okamžitou hodnotu vstupní proměnné s hodnotami dvou dalších vstupních proměnných - s dolní a horní mezní hodnotou. Výstupem komparátoru CMP může být jediná proměnná typu BOOL nebo dvojice proměnných BOOL. Ve druhém případě má první dvouhodnotový výstup význam indikace poklesu hodnoty testovaného vstupu pod dolní mez, druhý dvouhodnotový výstup má význam indikace nárůstu hodnoty testovaného vstupu nad horní mez.

Má-li funkce CMP jediný výstup, indikuje tento výstup vybočení testovaného vstupu mimo rozsah určený oběma mezemi.

Je-li některý výstup ve stavu "1", je hodnota dolní meze zvětšena a hodnota horní meze snížena o hodnotu hystereze, která je čtvrtým vstupním parametrem funkce CMP - je tedy "toleranční pásmo" v tomto stavu zúženo; po přechodu výstupu do stavu "0" je toleranční pásmo rozšířeno zpět na hodnoty mezi určené vstupními parametry bez jakékoli korekce.

Funkce CMP může být používána pro generaci výstražných resp. poruchových hlášení a realizaci regulátorů s dvoupolohovým nebo třípolohovým výstupem. Obě meze i hystereze mohou být za provozu seřizovány nebo algoritmičticky řízeny.

### Formát volání funkce CMP

Přípustné jsou dvě varianty volání funkce:

```
U, LoLimU, HiLimU, Hyst > CMP > Out
```

nebo

```
U, LoLimU, HiLimU, Hyst, > CMP > Low, High
```

(čárka za parametrem Hyst je v posledním případě významná - určuje, že funkce CMP bude mít dva výstupy).

### Vstupní a výstupní parametry

Typ testovaného vstupu U určuje typ, na který jsou konvertovány další vstupy. Komparace probíhají po nezbytných konverzích mezi proměnnými typu FIXP nebo FLTP.

Není-li zadán parametr LoLimU, je výsledek stejný jako by byla hodnota LoLimu(t) minimální možná pro daný typ, není-li zadán parametr HiLimU, je výsledek stejný jako pro maximální možnou hodnotu HiLimu(t).

Výstup, resp. výstupy, jsou vždy typu BOOL a má-li funkce korektně pracovat s nenulovou hysterezí, musí být výstup statickou proměnnou (implicitní nebo deklarovanou jako STAT BOOL).

### Zpracování dat ve funkci CMP

Pro komparátor s jedním výstupem:

```
IF (Out)
```

```
{ IF (((LoLimu(t)+Hyst(t))<<u(t)) & (u(t)<<(HiLimu(t)-Hyst(t))) 0 > Out(t)
}
```

```
ELSE IF (((u(t)<<LoLimu(t)) | (HiLimu(t)<<u(t))) 1 > Out(t);
```

Pro komparátor se dvěma výstupy:

```
IF (Low) { IF ((u(t) >> (LoLimu(t)+Hyst(t)) 0 > Low(t) }
```

```
ELSE IF ((u(t) << LoLimu(t)) 1 > Low(t);
```

```
IF (High) { IF ((u(t) << (HiLimu(t)-Hyst(t)) 0 > High(t) }
```

```
ELSE IF ((u(t) >> HiLimu(t)) 1 > High(t);
```

## 4.3 SEL - řízený výběr alternativních signálů

### Určení funkce a její charakteristika

Funkce SEL (selektce) je vlastně přepínačem - hodnota prvního vstupu typu BOOL určuje, zda je výstupní proměnné přiřazena momentální hodnota druhého nebo třetího vstupu.

### Formát volání funkce SEL

Funkce má pevný počet parametrů, tři vstupní a jeden výstupní:

```
B, U1, U2 > SEL > Y
```

### Vstupní a výstupní parametry

označení parametru	vnitřní reprezentace	typ parametru	význam parametru
B	B(t)	BOOL	řídící vstup
U1	u1(t)	FIXP FLTP BOOL	1. přepínaný vstup
U2	u2(t)	FIXP FLTP BOOL	2. přepínaný vstup
Y	y(t)	FIXP FLTP BOOL	výstup přepínače

Typ výstupu a 2. přepínaného vstupu je určen typem 1. přepínaného vstupu, přičemž BYTE vstupy se konvertují na FIXP.

### Zpracování dat ve funkci SEL

V jazyce LEDA lze funkci popsat takto (pro analogové vstupy):

```
IF (B(t))    u1(t) > y(t)
ELSE        u2(t) > y(t);
```

## 4.4 FIL - filtr 1. řádu

### Určení funkce a její charakteristika

Funkce FIL realizuje algoritmus číslicového zpětnovazebního filtru 1. řádu se z-přenosem:

$$Y(z)/U(z) = B \cdot z / (z - A)$$

kde:

$$B = T_v / \tau$$

$$A = 1 - T_v / \tau$$

a  $T_v$  je perioda zpracování sítě, ve které je uvažovaný "exemplář" funkce FIL zařazen.

Filtr FIL je dolní propustí - jde o diskretizovaný spojitý filtr s přenosem:

$$Y(p)/U(p) = 1 / (\tau \cdot p + 1)$$

Funkce FIL může být užívána pro potlačení nežádoucích šumových složek s frekvencí vyšší než je frekvence užitečného signálu a pro realizaci modelů řízených soustav pro simulační účely nebo i pro přímé řízení - např. v rámci tzv. Smithova prediktoru.

### Formát volání funkce FIL

Funkce má pevný počet parametrů:

```
U, Tv_Tau > FIL > Y
```

### Vstupní a výstupní parametry

označení parametru	typ parametru	vnitřní reprezentace	implicitní reprezentace	vnější reprezentace	fyzikální rozměr
U	FIXP	u(t)	u(t) < 80*Inp	Inp	[%]
Tv_Tau	FLTP	Tv_Tau(t)	Tv_Tau(t) < Tv/Tau	Tau	[s]
Y	STAT FIXP	y(t)	y(t) < 80*Outp	Outp	[%]

### Zpracování dat ve funkci FIL

Jestliže pracuje program v režimu "studený start", provedou všechny "exempláře" volání funkce FIL přiřazení:

```
u(t) > Y(t);
```

tedy nastavení počáteční hodnoty výstupu filtru na momentální hodnotu vstupního signálu.

Po ukončení studeného startu provádí FIL v každé periodě zpracování následující vyčíslení aritmetického výrazu a přiřazení:

```
(u(t) - y(t-1)) * Tv_Tau + y(t-1) > y(t);
```

hodnota  $y(t-1)$  je hodnotou jediné stavové veličiny funkce FIL - výstupní proměnná musí být statická, může tedy jít o implicitní proměnnou nebo o proměnnou deklarovanou jako STAT .... Jestliže uživatel nepoužije statickou proměnnou, zajistí statický výstup překladač a doplní případně další přiřazovací příkaz, který kopíruje hodnotu uživateli přímo nepřístupné statické proměnné do lokální proměnné uvedené v programu na místě parametru Y. V

tomto případě však odpadá možnost sledovat přímo na výstupu funkce FIL momentální hodnoty a není také možné ruční nastavení hodnoty výstupu filtru (vnucení počáteční hodnoty výstupu).

Nižší řády výstupu filtru jsou uchovávány ve statické oblasti paměti, kterou překladač vždy alokuje pro každý exemplář filtru FIL. Přesnější uchovávání starší hodnoty výstupu je významné pro extrémně malé hodnoty  $T_v/T_\tau$ , tj. velký poměr  $T_\tau/T_v$ .

Časová konstanta filtru může být zadána jako skutečná konstanta, nebo může jít o inicializovanou proměnnou - skalární parametr, který dává možnost seřizování za běhu (bez nového překladu). Jinou možností je na místě parametru  $T_v/T_\tau$  použít proměnnou, která může být výstupem části uživatelského programu řídicího spojitě nebo skokově dynamiku filtru.

## 4.5 SUM - sumace časové řady

### Určení funkce a její charakteristika

Funkce SUM provádí sumaci časové řady, tzn., že k aktuální hodnotě výstupu  $y(t)$  přičítá v každé periodě zpracování momentální hodnotu vstupu  $u(t)$ . Tento S-člen může být, stejně jako PSD regulátor realizovaný funkcí PID, převeden prostřednictvím dvouhodnotového vstupu  $B_i(t)$  do režimu sledování, kdy je hodnotou dalšího vstupu (žádané počáteční hodnoty  $y_i(t)$ ) přepisován výstup  $y(t)$ . Na S-člen lze pohlížet jako na diskretizovaný integrátor (integrace obdélníkovou metodou).

Funkce SUM má mnohostranné užití jak v algoritmech přímého řízení (např. pro vyhodnocování integrálních kritérií kvality řídicích pochodů při jejich optimalizaci), tak i pro účely bilanční a statistické (odhady celkového množství surovin, energie a výstupních produktů, odhady středních hodnot, disperzí resp. směrodatných odchylek různých technologických nebo odvozených veličin).

### Vyvolání funkce SUM

Formát je pevný, se třemi vstupními a jedním výstupním parametrem:

```
U, Bi, Yi > SUM > Y
```

### Parametry funkce

Typ parametru U určuje typ parametrů  $Y_i$  a Y alternativně jako FIXP nebo FLTP. Vstup  $B_i$  je typu BOOL a určuje, zda funkce pracuje v režimu inicializace (sledování) - pro  $B_i(t) == 0$ , nebo vykonává sumaci vstupní posloupnosti.

### Zpracování dat ve funkci SUM

V jazyce LEDA:

```
IF (Studený_start | Bi(t)==0)   yi(t) > y(t)
ELSE    u(t) + y(t-1), Min, Max > SAT > y(t);
```

kde Min resp. Max je minimální resp. maximální přípustná hodnota proměnné daného typu a SAT je standardní funkce saturace.

## 4.6 PID regulátor

### Určení funkce a její základní charakteristika

Funkce PID může sloužit jako ústřední člen regulátorů s chováním I, PI a PID - ovšem v diskretizované verzi, tj. s kanálem sumačním S, proporcionálním P a diferenčním D, tedy přesněji řečeno - jako S, PS a PSD regulátory.

Regulátory realizovatelné pomocí funkce PID musí mít vždy astatické chování (nulový pól) - integrační, (resp.sumační) složka musí mít nenulové zesílení.

Regulátory konstruované pomocí funkce PID mohou zahrnovat pomocnou dopřednou vazbu, filtraci akčních zásahů (výstupu regulátoru) dolnofrekvenční propustí 1.řádu (útlum 20 dB na dekádu), omezení přírůstků výstupu (limitaci rychlosti) a omezení velikosti výstupu (limitaci polohy).

Regulátory mohou pracovat ve dvou provozních režimech - režimu "sledování", ve kterém výstup regulátoru sleduje hodnotu jednoho ze vstupů označovaného dále jako žádaná hodnota pro sledování a režimu "automatického řízení", ve kterém je výstup regulátoru určován na základě posloupnosti žádaných hodnot regulované veličiny, posloupnosti skutečných hodnot regulované veličiny a posloupností hodnot všech seřiditelných parametrů regulátoru. Funkce PID regulátoru tedy může zajistit beznárazové přepínání do režimu automatického řízení.

Konstrukce regulátoru - funkce - PID podle tzv. rychlostního algoritmu vylučuje "windup" polohy (výstupu) - jev, při kterém saturovanému výstupu neodpovídá neadekvátně vysoká popř. nízká hodnota výstupu integrátoru resp. sumačního členu, tedy jedné ze stavových veličin regulátoru pojmávaného jako dynamický systém.

Funkce PID umožňuje měnit za provozu seřízení všech parametrů (včetně mezí pro saturace rychlosti a polohy), a to jak ručními zásahy seřizovače, tak i algoritmicky, umožňuje tedy i konstrukci některých variant adaptivních regulátorů.

### Důležité upozornění

Při seřizování parametrů integračního a derivačního kanálu v prostředí COLEDA je vhodné používat skalární parametry s implicitními reprezentacemi (označení \$ bez identifikátoru reprezentace), je však třeba si uvědomit, že hodnoty parametrů zpracovávané v exekutivě funkce PID jsou v těchto reprezentacích určovány jako závislé nejen na periodě zpracování dat v síti, ale také na parametru zesílení. Změna celkového zesílení vyvolává automaticky i změnu zesílení v I a D kanálu a tím i změny zpětně určovaných parametrů  $T_i$ ,  $T_d$ . Tomuto jevu lze snadno zabránit - jestliže jsou znaky komunikačního přiřazení v okamžiku stisku klávesy ENTER v poloze pro zadávání "<" u všech tří jmenovaných parametrů a kurzor je mimo jakékoli zadávací okénko provede se výpočet všech tří zesílení a přenos všech nových hodnot seřiditelných parametrů dané obrazovky do řídicí stanice.

### Vyvolání funkce PID

Vyvolání funkce PID má následující formát s pevným počtem čtrnácti vstupních parametrů a jedním výstupním parametrem:

SP, PV, Aux, AUT, FOLL.SP, GAIN, Ki, Kd, Kaux, Tv-Tau,  
LoLimDY, HiLimDY, LoLimY, HiLimY > PID > Y

### Vstupní a výstupní parametry funkce PID

označení veličiny	typ parametru	označení parametru	komentář v příkladu	význam parametru
w(t)	FIXP	SP	SETPOINT	žádaná hodnota regulované veličiny
x(t)	FIXP	PV	PROC.VAR	skutečná hodnota regulované veličiny
v(t)	FIXP	Aux	AUX.VAR	hodnota pomocného vstupního signálu
A(t)	BOOL	AUT	AUT FOLL	řízení režimu: automaticky sledování
B(t)	FIXP	FOLL.SP	FOLL.SP	žádaná hodnota pro režim sledování
Kc(t)	FLTP	GAIN	GAIN	celkové zesílení regulátoru
Ki(t)	FLTP	Ki	Ti	zesílení integrační složky – integrační časová konstanta
Kd(t)	FLTP	Kd	Td	zesílení derivační složky - derivační časová konstanta
Kv(t)	FLTP	Kaux	Kaux	zesílení pomocného vstupního signálu - dopředné vazby
Tv_Tau(t)	FLTP	Tv_Tau	Tau	časová konstanta filtru 1.řádu
LoLimdy(t)	FIXP	LoLimDY	LoLimDY	dolní mezní rychlost výstupu regulátoru
HiLimdy(t)	FIXP	HiLimDY	HiLimDY	horní mezní rychlost výstupu regulátoru
LoLimy(t)	FIXP	LoLimY	LoLimY	dolní mezní hodnota výstupu regulátoru
HiLimy(t)	FIXP	HiLimY	HiLimY	horní mezní hodnota výstupu regulátoru
y(t)	STAT FIXP	Y	Y	výstup regulátoru (akční veličina)

## Reprezentace hodnot parametrů

označení parametru vnitřní reprezentace	implicitní reprezentace	označení řetězce vnější reprezentace	fyzikální rozměr
w(t)	w(t) < 80*SETPOINT	SETPOINT	[%]
x(t)	w(t) < 80*PROC.VAR	PROC.VAR	[%]
v(t)	w(t) < 80*AUX.VAR	AUX.VAR	[%]
A(t)	A(t) < 0   A(t) < 1	AUT FOLL	-
B(t)	B(t) < 80*FOLL.SP	FOLL.SP	[%]
Kc(t)	Kc(t) < 256*GAIN	GAIN	[1]
Ki(t)	Ki(t) < Kc(t)*Tv/Ti	Ti	[s]
Kd(t)	Kd(t) < Kc(t)*Td/Tv	Td	[s]
Kv(t)	Kv(t) < 256*Kaux	Kaux	[1]
Tv_Tau(t)	Tv_Tau < Tv/Tau	Tau	[s]
LoLimdy(t)	LoLimdy(t) < 80*Tv*LoLimDY	LoLimDY	[%/s]
HiLimdy(t)	HiLimdy(t) < 80*Tv*HiLimDY	HiLimDY	[%/s]
LoLimy(t)	LoLimy(t) < 80*LoLimDY	LoLimY	[%]
HiLimy(t)	HiLimy(t) < 80*HiLimDY	HiLimY	[%]

Symbol Tv přitom označuje periodu zpracování (vzorkování) sítě, ve které je daná funkce PID zařazena (s rozměrem [s]).

Jestliže není parametr GAIN zadán, dává implicitní reprezentace  $0 > Kc(t)$ , ale určuje  $Ki(t) < 256 * Tv / Ti$ , tj. jako pro jednotkové zesílení.

## Zpracování dat ve funkci PID

Jestliže platí:

$$A(t) == 0$$

nebo je program v režimu "studený start",

provede funkce PID přiřazení:

$$B(t) > y(t);$$

což znamená, že PID pracuje v režimu sledování - výstup y(t) sleduje žádanou hodnotu pro sledování.

Jestliže platí:

$$A(t) == 1$$

pracuje PID v režimu automatického řízení, kdy jsou v každé periodě zpracování vstupních parametrů provedena tato přiřazení:

$$w(t) - x(t) > e(t); \quad /*regulační odchylka*/$$

$$e(t) - e(t-1) > de(t); \quad /*1.diference regulační odchylky*/$$

$$de(t) - e(t-1) + e(t-2) > d2e(t); \quad /*2.diference regulační odchylky*/$$

následuje vyčíslení výstupu "ideálního DPD<sup>2</sup> regulátoru":

$$Kc(t) * de(t) + Ki(t) * e(t) + Kd(t) * d2e(t) + Kv(t) * v(t) > dyi(t);$$

a filtrace 1.řádu:

$$(dyi(t) - dy(t-1)) * Tv\_Tau + dy(t-1) > dy(t);$$

dále následuje oboustranná saturace rychlosti výstupu regulátoru (saturace 1.diference signálu y(t)):

$$LoLimdy(t), dy(t) > \underline{MAX}, HiLimdy(t) > \underline{MIN} > dy(t);$$

dalším krokem zpracování dat je integrace, resp. sumace, posloupnosti dy(t):

$$dy(t) + y(t-1) > y(t);$$

na závěr je provedena oboustranná saturace výstupu:

$LoLimy(t), y(t) > \underline{MAX}, HiLimy(t) > \underline{MIN} > y(t);$

kterou nejvýznamnější kroky zpracování dat funkcí PID v režimu automatického řízení končí.

Veličiny:  $e(t-1), e(t-2), dy(t-1)$  a  $y(t-1)$

jsou stavovými veličinami dynamického systému, kterým funkce PID je. Kromě výstupu  $y(t)$ , který musí být z tohoto důvodu do příštího vyvolání stejného "exempláře" funkce PID fixován jako hodnota statické proměnné, jsou ostatní stavové veličiny každého regulátoru zachyceny v jedinečném, uživateli přímo nepřístupném, stavovém vektoru určitého regulátoru. Jestliže uživatel nepoužije na místě výstupního parametru Y statickou proměnnou, doplní ji překladač jako uživatelsky nepřístupnou buňku a za volání funkce PID doplní přiřazovací příkaz, který výstup regulátoru přepisuje do proměnné, kterou použil v programu uživatel. Nastane-li popsaná situace, nelze výstup regulátoru zobrazovat a nelze mu také komunikačně přiřazovat "vnucené hodnoty".

### Příklad komentovaného vyvolání funkce PID

U funkce PID lze považovat za vhodné formátovat volání do tvaru formuláře, např. podle tohoto vzoru:

```

/*=====*/
/*SETPOINT*/ TV123W           , /* zadana hodnota teploty */
/*PROC.VAR*/ TV123X           , /* skutecna teplota */
/*AUX.VAR.*/                  , /*
/*AUT|FOLL*/ AUT123           , /* povel "sleduj"pro "MAN" */
/*FOLL.S.P*/ PV123X           , /* skutecna poloha ventilu */
/* GAIN      Ti      Td      Kaux      Tau      */
F00<$ 1.0000, F01<$ 999.99s, F02<$ 000.00s, F03<$ 0.0000, F04<$ 000.00s,
/* LoLimDY      HiLimDY      LoLimY      HiLimY      */
P00<$ -100.0%/s, P01<$ 100.0%/s, P02<$ 000.0%, P03<$ 100.0%
> PID > PV123Y ;           /* zadana h.polohy ventilu topne vody */
/*=====*/

```

## 4.7 PDC regulátor

### Určení funkce a její základní charakteristika

Funkce PDC může sloužit jako ústřední člen regulátorů s chováním P, D a PD - ovšem v diskretizované verzi, tj. s kanálem proporcionálním P a diferenčním D.

Regulátory konstruované pomocí funkce PDC mohou zahrnovat filtraci akčních zásahů (výstupu regulátoru) dolnofrekvenční propustí 1. řádu (útlum 20 dB na dekádu) a omezení velikosti výstupu (limitaci polohy).

Funkce PDC umožňuje měnit za provozu seřízení všech parametrů (včetně mezí pro saturaci polohy), a to jak ručními zásahy seřizovače, tak i algoritmicky, umožňuje tedy i konstrukci různých variant adaptivních regulátorů.

### Vyvolání funkce PDC

Vyvolání funkce PDC má následující formát s pevným počtem sedmi vstupních parametrů a jedním výstupním parametrem:

$SP, PV, GAIN, Kd, Tv\_Tau, LoLimY, HiLimY > PDC > Y$

### Vstupní a výstupní parametry funkce PDC

označení veličiny	typ parametru	označení parametru	komentář v příkladu	význam parametru
w(t)	FIXP	SP	SETPOINT	žádaná hodnota regulované veličiny

x(t)	FIXP	PV	PROC.VAR	skutečná hodnota regulované veličiny
Kc(t)	FLTP	GAIN	GAIN	celkové zesílení regulátoru
Kd(t)	FLTP	Kd	Td	zesílení derivační složky - derivační časová konstanta
Tv_Tau(t)	FLTP	Tv_Tau	Tau	časová konstanta filtru 1. řádu
LoLimy(t)	FIXP	LoLimY	LoLimY	dolní mezní hodnota výstupu regulátoru
HiLimy(t)	FIXP	HiLimY	HiLimY	horní mezní hodnota výstupu regulátoru
y(t)	STAT FIXP	Y	Y	výstup regulátoru (akční veličina)

### Reprezentace hodnot parametrů

označení parametru vnitřní reprezentace	implicitní reprezentace	označení řetězce vnější reprezentace	fyzikální rozměr
w(t)	w(t) < 80*SETPOINT	SETPOINT	[%]
x(t)	w(t) < 80*PROC.VAR	PROC.VAR	[%]
Kc(t)	Kc(t) < 256*GAIN	GAIN	[1]
Kd(t)	Kd(t) < Kc(t)*Td/Tv	Td	[s]
Tv_Tau(t)	Tv_Tau < Tv/Tau	Tau	[s]
LoLimy(t)	LoLimy(t) < 80*LoLimDY	LoLimY	[%]
HiLimy(t)	HiLimy(t) < 80*HiLimDY	HiLimY	[%]

Symbol Tv přitom označuje periodu zpracování (vzorkování) sítě, ve které je daná funkce PDC zařazena (s rozměrem [s]).

Jestliže není parametr GAIN zadán, dává implicitní reprezentace  $0 > Kc(t)$ , a  $Kd(t) < 256 * Td / Tv$ .

### Zpracování dat ve funkci PDC

V každé periodě zpracování vstupních parametrů jsou provedena tato přiřazení:

$$w(t) - x(t) > e(t); \quad /*regulační odchylka*/$$

$$e(t) - e(t-1) > de(t); \quad /*1.diference regulační odchylky*/$$

následuje vyčíslení výstupu "ideálního PD regulátoru":

$$Kc(t) * e(t) + Kd(t) * de(t) > yi(t);$$

a filtrace 1.řádu:

$$(yi(t) - y(t-1)) * Tv\_Tau + y(t-1) > y(t);$$

dále následuje oboustranná saturace výstupu regulátoru:

$$LoLimy(t), y(t) > \underline{MAX}, HiLimy(t) > \underline{MIN} > y(t);$$

tím nejvýznamnější kroky zpracování dat funkcí PDC končí.

Veličiny: e(t-1) a y(t-1) jsou stavovými veličinami dynamického systému, kterým funkce PDC je.

### Příklad komentovaného vyvolání funkce PDC

U funkce PDC lze považovat za vhodné formátovat volání do tvaru formuláře, např. podle tohoto vzoru:

```
/*=====*/
/*SETPOINT*/ L456W , /* zadana hodnota hladiny */
/*PROC.VAR*/ L456X , /* skutecna h. hladiny */
/* GAIN Td Tau */
F00<$ 1.0000, F01<$ 999.99s, F02<$ 000.00s,
/* LoLimY HiLimY */
P00<$ 000.0%, P01<$ 100.0%
> PDC > F456W ; /* zadana h.prutoku */
/*=====*/
```

## 4.8 SET, RES - nahození, resp. shození, dvouhodnotové proměnné

### Určení funkcí a jejich charakteristika

Funkce SET a RES umožňují realizovat klopný obvod (flipflop). Obě funkce mají jediný vstup typu BOOL a jeden nebo několik (nejvýše však 16) výstupů stejného typu. K jediné výstupní proměnné může v programu existovat libovolný počet funkcí SET a RES.

### Vyvolání funkcí SET a RES

Volání SET:

```
B > SET > X resp. B > SET > X,Y,Z
```

Volání RES:

```
B > RES > X resp. B > RES > X,Y,Z
```

U obou funkcí jsou výstupními parametry zpravidla proměnné deklarované jako STAT BOOL nebo implicitní proměnné BOOL, není to však podmínkou. Je-li výstupem lokální automatická proměnná, nelze zobrazovat její hodnotu, nelze ji ručními zásahy modifikovat a její význam může být pouze krátkodobý - nemůže jít v pravém smyslu slova o realizaci klopného obvodu jako relativně trvalé dvouhodnotové paměti.

### Zpracování dat ve funkcích SET a RES

SET:

```
IF (B(t)) 1 > x(t); resp. IF (B(t)) { 1>X; 1>Y; 1>Z };
```

RES:

```
IF (B(t)) 0 > x(t); resp. IF (B(t)) { 0>X; 0>Y; 0>Z };
```

### Pozor!

Vzhledem ke skutečnosti, že funkce SET a RES zapisují do výstupní proměnné jen při pravdivé podmínce B, nesmí být jejich výstupním parametrem indexovaná proměnná (prvek BOOL pole) s proměnným indexem!

Rovněž použití ve výrazech je problematické – uživatelsky nepřístupná schránka mezivýsledku bude podmíněně nahozena bez možnosti shození.

## 4.9 DSR, SDR - zpožděné nahození, resp. shození, BOOL výstupu

### Určení funkcí a jejich charakteristika

Funkce DSR zpožďuje náběžnou hranu BOOL výstupu proti náběžné hraně BOOL vstupu o zadaný časový interval Delay; sestupná hrana výstupu sleduje vstup.

Funkce SDR zpožďuje sestupnou hranu BOOL výstupu proti sestupné hraně BOOL vstupu o zadaný časový interval Delay; náběžná hrana výstupu sleduje vstup.

Zpoždění Delay může být za chodu seřizováno nebo programově řízeno. Obě funkce se mohou uplatňovat při konstrukci sekvenčních automatů, např. v obvodu kontroly splnění povelu v časovém limitu a kontroly validity zpětných hlášení z technologie.

### Vyvolání funkcí DSR a SDR

Volání DSR :

```
B, Delay > DSR > X
```

Volání SDR :

```
B, Delay > SDR > X
```

### Vstupní a výstupní parametry

označení parametru	vnitřní reprezentace	typ parametru	implicitní reprezentace	vnější reprezent.	fyzikální rozměr
B	B(t)	BOOL	B(t)<0   B(t)<1	"0" "1"	-
Delay	Delay(t)	FIXP	Delay(t)<Tau/Tv	Tau	[s]
X	x(t)	STAT BOOL	x(t)<0   x(t)<1	"0" "1"	-

Vedle statické proměnné výstupu x(t) mají funkce DSR resp. SDR další stavovou proměnnou nepřístupnou přímo uživateli, která zachycuje aktuální hodnotu zpoždění vůči náběžné resp. sestupné hraně vstupu jako počet period zpracování dané sítě délky Tv (proměnná Citac).

### Zpracování dat ve funkcích DSR a SDR

DSR v jazyce LEDA :

```
IF (!x(t-1))
    IF (B(t)) /* x(t-1)==0 */
        { Citac + 1 > Citac;
          IF (Citac >> Delay(t) { 1 > x(t); 0 > Citac; }
        }
    ELSE 0 > Citac
ELSE
    { 0 > Citac; /* x(t-1)==1 */
      IF (!B(t)) 0 > x(t);
    };
```

SDR v jazyce LEDA :

```

IF (x(t-1))
    IF (!B(t))                                /* x(t-1)==1 */
        { Citac + 1 > Citac;
          IF (Citac >> Delay(t) { 0 > x(t); 0 > Citac; }
        }
    ELSE 0 > Citac
ELSE
    { 0 > Citac;                                /* x(t-1)==0 */
      IF (B(t)) 1 > x(t);
    };

```

## 4.10 IMP, NIMP - impuls od náběžné hrany, impuls od sestupné hrany

### Určení funkce a její charakteristika

Funkce IMP generuje na svém dvouhodnotovém výstupu impuls délky  $T_v$  vždy při přechodu BOOL vstupu ze stavu "0" do stavu "1", funkce NIMP generuje impuls při přechodu vstupu z "1" do "0".

### Vyvolání funkce IMP nebo NIMP

Obě funkce mají jediný vstup a jediný výstup:

```
B > IMP > X
```

nebo

```
B > NIMP > X
```

### Parametry funkce

Vstup  $B(t)$  i výstup  $x(t)$  jsou typu BOOL; může jít jak o statické proměnné, tak i o lokální automatické proměnné.

### Zpracování dat ve funkci IMP

```

IF (B(t) & !B(t-1)) 1 > x(t)
ELSE                  0 > x(t);

```

## 4.11 DIF - 1. difference resp. indikace změny

### Určení funkce a její charakteristika

Funkce DIF je použitelná jak pro analogové vstupy - pak je i její výstup analogový a je tvořen posloupností 1. difference vstupní posloupnosti, tak i pro dvouhodnotové vstupy, kdy funkce DIF vyhodnocuje všechny změny vstupu ze stavu "0" do stavu "1" a naopak, a pro každý výskyt změny generuje na výstupu impuls délky  $T_v$ .

Funkce DIF může být použita (úspěšněji než funkce PDC) pro určování rychlosti změn analogových veličin, např. při kontrole validity signálů přijímaných z technologického procesu, v dopředných vazbách regulátorů, při přípravě dat pro identifikaci parametrů inkrementálních modelů a pod..

Dvouhodnotová verze funkce DIF má velmi široké užití např. při generování vstupů pro nahazování a shazování klopných obvodů, při spouštění procesů iniciovaných zásahem operátora, při vyhodnocování počtu sepnutí akčních orgánů, ... .

### Vyvolání funkce DIF

Pro analogové vstupy:

`U > DIF > Y`

Pro dvouhodnotové vstupy:

`B > DIF > X`

### Parametry funkce DIF

Pro analogové vstupy:

označení parametru	vnitřní reprezentace	typ parametru
U	u(t)	FIXP FLTP
Y	y(t)	FIXP FLTP

Pro dvouhodnotové vstupy:

označení parametru	vnitřní reprezentace	typ parametru
B	b(t)	BOOL
X	x(t)	BOOL

### Zpracování dat ve funkci DIF

Pro analogové vstupy:

`u(t) - u(t-1), Min, Max > SAT > y(t);`

přičemž Min a Max označují minimum a maximum povolené pro daný typ proměnné.

Pro dvouhodnotové vstupy:

`IF (b(t) != b(t-1)) 1 > x(t)  
ELSE 0 > x(t);`

Každý "exemplář" funkce DIF má překladačem přidělenou uživateli nepřístupnou stavovou veličinu u(t-1) resp. b(t-1) daného typu.

## 4.12 MSF - monostabilní klopný obvod

### Určení funkce a její charakteristika

Náběžná hrana výstupu sleduje náběžnou hranu vstupu; délka výstupního impulsu je určena druhým vstupním parametrem Delay. Požadovaná délka impulsu může být za běhu měněna. Výstupní impuls nemůže být zkrácen působením vstupu pod délku danou momentální hodnotou parametru Delay, může však být prodloužen novou náběžnou hranou vstupu.

### Vyvolání funkce MSF

`B, Delay > MSF > X`

### Parametry funkce MSF

označení parametru	vnitřní reprezentace	typ parametru	implicitní reprezentace	vnější reprezentace	fyzikální rozměr
B	b(t)	BOOL	b(t)<0   b(t)<1	"0" "1"	-
Delay	Delay(t)	FIXP	Delay(t)<Tau/Tv	Tau	[s]
X	x(t)	STAT BOOL	x(t)<0   x(t)<1	"0" "1"	-

### Zpracování dat ve funkci MSF

```
IF (!b(t)) { 0 > Priznak(t); /* vstup je == 0 */
    IF (Citac >= Delay(t))
```

```

        { 0 > Citac;
          0 > x(t)
        }
      ELSE Citac + 1 > Citac
    }
ELSE { IF (Priznak(t-1))                               /* vstup je == 1 */
      { IF (Citac >= Delay(t))
        { 0 > Citac;
          0 > x(t)
        }
      ELSE Citac + 1 > Citac
    }
  ELSE { 1 > Priznak(t);                               /* nabezna hrana */
        0 > Citac;
        1 > x(t)
      }
};

```

Veličiny Priznak(t-1) a Citac(t-1) tvoří stavový vektor funkce MSF. Starší hodnota výstupu x(t-1) sice v těle funkce nevystupuje, ale měli bychom si uvědomit, že výstup není nastavován nebo nulován v každé periodě - proto na místě výstupu tohoto klopného obvodu prakticky vždy použijeme statickou proměnnou.

## 4.13 NLT - nelineární transformace

### Určení funkce a její charakteristika

Funkce NLT má velmi široké užití - může sloužit pro kompenzaci nelinearit čidel, akčních orgánů i vlastních řízených procesů, umožňuje programovat změny různých parametrů (např. zesílení v jednotlivých kanálech regulátorů) v závislosti na měřených veličinách nebo vyhodnocovaných kritériích, může simulovat chování různých subsystémů řízených procesů, ... .

Funkce NLT provádí statickou nelineární transformaci analogového vstupního signálu na signál výstupní - příslušná charakteristika se zadává tabulkou souřadnic bodů zlomu po úsecích lineární funkce. Mezi sousedními body zlomu probíhá tedy lineární interpolace; za krajními body zlomu nastává saturace výstupu. Všechny souřadnice bodu zlomu jsou obecně proměnné za běhu programu - statickou charakteristiku lze tedy modifikovat jak ručními zásahy, tak i algoritmi.

### Vyvolání funkce NLT

Funkce NLT nemá pevně určený počet vstupních parametrů - uživatel zvolí počet bodů zlomu N a zadá 1+2N vstupních parametrů:

```
U, X1, Y1, X2, Y2, ..., XN, YN > NLT > Y
```

Překladač omezuje počet bodů zlomu na  $N \leq 50$ , a to pouze z důvodu omezení nároků na čas procesoru.

### Parametry funkce NLT

Je-li vstup U nebo alespoň jedna souřadnice Xi nebo Yi typu FLTP proběhne zpracování dat s tímto typem a na výstupu se provede případná konverze na FIXP popř. BYTE, v opačném případě proběhne zpracování ve FIXP. V případě, že je vstup U typu FIXP a všechny souřadnice Xi, Yi jsou konstantami FIXP, připraví překladač pro exekutivu funkce NLT datovou strukturu, která obsahuje mj. i směrnice všech lineárních úseků; v ostatních případech

se při interpretaci zpracovávají přímo souřadnice bodů zlomu. Pro konstantní FIXP souřadnice je přesnost lineární interpolace omezena malou rozlišovací schopností pamatované směrnice (šestnáctibitový údaj) !

Pro souřadnice  $x_i(t)$  musí platit:  $x_1(t) \leq x_2(t) \leq \dots \leq x_N(t)$ !

### Zpracování dat ve funkci NLT

Zpracování dat je zahájeno vyhledáním úseku, pro který platí:

$$(x_i(t) \leq u(t)) \ \& \ (u(t) \leq x_{i+1}(t))$$

při vyhledávání úseku se přitom postupuje od vyšších indexů i směrem k nižším.

Funkce NLT připouští v krajním případě i směrnice s absolutní hodnotou nekonečné velikosti - úseky pro které platí:

$$x_i(t) == x_{i+1}(t) \ \text{a} \ y_i(t) \neq y_{i+1}(t)$$

jsou zpracovány tak, že pro:

$$u(t) == x_i(t)$$

přiřadí NLT:

$$y_{i+1}(t) > y(t);$$

Nenastal-li tento zvláštní případ, vyhodnotí se následující aritmetický výraz:

$$(u(t) - x_i(t)) * (y_{i+1}(t) - y_i(t)) / (x_{i+1}(t) - x_i(t)) + y_i(t) > y(t);$$

Mimo rozsah  $\langle x_1(t), x_N(t) \rangle$  nastává saturace:

$$\text{IF } (u(t) >> x_N(t)) \ y_N(t) > y(t);$$

$$\text{IF } (u(t) << x_1(t)) \ y_1(t) > y(t);$$

### Příklad vyvolání funkce NLT:

```
(U15 > $ _____), 000.0% $, 000.0% $,
015.0% $, 023.5% $,
060.0% $, 087.3% $,
080.0% $, 100.0% $
NLT > KorU15 > $ _____;
```

## 4.14 CNT - čítač

### Určení funkce a její charakteristika

Funkce CNT je vratným čítačem s možností přednastavení výstupu. Užití čítačů je velmi široké - např. pro vyhodnocování počtu událostí indikovaných impulsy nebo vyhodnocování celkové doby trvání určitého stavu, a to jak pro účely přímého řízení (např. dávkování tekutin na základě údajů čidel průtoku s impulsním výstupem), tak i pro účely diagnostické a pro optimalizační řídicí algoritmy (např. vyhodnocování počtu najetí určitých technologických zařízení nebo doby jejich provozu v daném režimu a příp. navazující optimalizované odstavení a najíždění). Počet průchodů větvi programu lze určovat také funkcí CNT.

### Volání funkce CNT

$$CU, CD, Bi, Yi > CNT > Y$$

### Parametry funkce

označení parametru	vnitřní reprezentace	typ parametru	význam parametru
CU	Cu(t)	BOOL	čítej vzestupně
CD	Cd(t)	BOOL	čítej sestupně

Bi	Bi(t)	BOOL	nastav výstup
Yi	yi(t)	FIXP	h.nastavení výstupu
Y	y(t)	FIXP	výstup čítače

Kterýkoli ze vstupních parametrů může zůstat nezadán, je-li však zadán parametr Bi, musí být zadán i parametr Yi.

#### Zpracování dat ve funkci CNT

```
IF (Bi(t)) yi(t) > y(t);
IF (Cd(t)) IF (y(t)==0) 32767 > y(t)
            ELSE y(t-1) - 1 > y(t);
IF (Cu(t)) IF (y(t)==32767) 0 > y(t)
            ELSE y(t-1) + 1 > y(t);
```

Jestliže není zadán parametr Bi, nepřichází nastavení výstupu v úvahu, není-li zadán parametr Cd, není možná dekrementace čítače. Čítač inkrementuje nejen pro Cu(t)≠0, ale i v případě, že parametr Cu nebyl vůbec zadán.

Jak je z uvedeného fragmentu programu zřejmé, pracuje čítač modulo 2<sup>15</sup>.

Nedodržení výstupního typu FIXP se může nežádoucím způsobem projevit v případech, kdy je výstupní proměnná modifikována také mimo volání CNT! Překladač sice zajistí vždy statickou proměnnou správného typu na výstupu funkce a vygeneruje navazující konverzi, ale případná modifikace finální proměnné se nemůže projevit na uživateli nepřístupném výstupu!

## 4.15 DBN - pásmo necitlivosti

#### Určení funkce a její charakteristika

Funkce DBN potlačuje vliv vstupního signálu, jestliže jeho hodnota leží v rozsahu <LoLimU(t), HiLimU(t)> - mezních hodnot určených 2. a 3. vstupním parametrem funkce. Meze pásma necitlivosti lze za běhu řídit resp. seřizovat.

Pásmo necitlivosti lze např. předřazovat regulátorům a omezovat tak frekvenci akčních zásahů resp. změn akčního zásahu.

#### Volání funkce DBN

```
U, LoLimU, HiLimU > DBN > Y
```

#### Parametry funkce

Typ proměnné U určuje typy dalších dvou parametrů a výstupu Y alternativně jako FIXP nebo FLTP.

#### Zpracování dat ve funkci DBN

```
IF (u(t) << LoLimU(t)) u(t) - LoLimU(t) > y(t)
ELSE IF(u(t) >> HiLimU(t)) u(t) - HiLimU(t) > y(t)
ELSE 0 > y(t);
```

## 4.16 SAT - saturace

#### Určení funkce a její charakteristika

Funkce SAT provádí oboustranné omezení vstupního signálu do rozsahu <LoLimy(t), HiLimy(t)>. Oba limity mohou být za běhu seřizovány nebo algoritmicky řízeny.

Saturace se používá např. pro omezování vlivnosti různých složek regulátorů s dopřednými vazbami, omezování povoleného rozsahu adaptovaných parametrů a omezování výstupů těch číslicových korekčních členů, které nejsou zakončeny některou ze standardních funkcí regulátorů PID, PDC nebo PDD. U funkce SAT má v některých případech smysl ztotožnění vstupní a výstupní proměnné.

### Volání funkce SAT

```
U, LoLimY, HiLimY > SAT > Y
```

### Parametry funkce

Typ parametru U určuje typy dalších dvou parametrů a výstupu Y alternativně jako FIXP nebo FLTP.

### Zpracování dat ve funkci SAT

```
IF      (u(t) << LoLimy(t))      LoLimy(t) > y(t)
ELSE IF(u(t) >> HiLimy(t))      HiLimy(t) > y(t)
ELSE                                     u(t) > y(t);
```

## 4.17 MAV - klouzavý průměr

### Určení funkce a její charakteristika

Funkce MAV vyhodnocuje aritmetický průměr z N nejaktuálnějších po sobě následujících vzorků vstupního signálu.

Klouzavý průměr se používá převážně při zpracování signálů pro zobrazování, archivace a tisk v operátorské resp. dispečerské stanici.

### Vyvolání funkce MAV

```
U, N > MAV > Y
```

### Parametry funkce

Typ parametru U určuje typ výstupu Y alternativně jako FIXP nebo FLTP. Druhý vstupní parametr N musí být konstanta typu FIXP - určuje délku cyklického zásobníku, který překladač vyhradí v oblasti statických proměnných (stavových vektorů dynamických funkcí).

### Zpracování dat ve funkci MAV

```
IF (Studený_start) FOR (1>i; i<=N; i+1>i) u(t) > Data[i];
Index + 1 > Index;
IF (Index >> N) 1 > Index;
u(t) > Data[Index];
0 > Suma;
FOR (1>i; i<=N; i+1>i) Data[i] + Suma > Suma; Suma/N > y(t);
```

Pole Data[1..N] tvoří cyklický zásobník, aktuální místo pro zápis určuje stavová proměnná Index.

Z uvedeného programu je zřejmé, že v režimu studeného startu je celý zásobník jednorázově naplněn aktuální hodnotou vstupu u(t), která tvoří také počáteční hodnotu vystupujícího klouzavého průměru.

## 4.18 MMA resp. MMI - maximum resp. minimum na klouzavém intervalu

### Určení funkcí a jejich charakteristika

Funkce MMA resp. MMI určuje hodnotu maxima resp. minima z posledních N vzorků vstupního signálu a jako druhý výstupní parametr poskytuje index nalezeného extrému, tj. "stáří" extrému vyjádřené počtem period zpracování Tv.

Obě funkce jsou určeny zejména pro zpracování dat v operátorské stanici jako přípravu pro zobrazování, archivace nebo tisk - provádějí (stejně jako funkce MAV) kompresi dat.

#### Volání funkce

```
U, N > MMA > Y, Index
```

resp.

```
U, N > MMI > Y, Index
```

#### Parametry funkcí

Typ parametru U určuje typ výstupu Y alternativně jako FIXP nebo FLTP. Druhý vstupní parametr N musí být konstanta typu FIXP - určuje délku cyklického zásobníku, který překladač vyhradí v oblasti statických proměnných (stavových vektorů dynamických funkcí).

#### Zpracování dat ve funkcích MMA a MMI

Funkce MMA:

```
IF (Studený_start) FOR (1>i; i<=N; i+1>i) u(t) > Data[i];
Index + 1 > Index;
IF (Index >> N) 1 > Index;
u(t) > Data[Index];
MinVar > Min;
FOR (1>i; i<=N; i+1>i) IF (Data[i]>Min) {Data[i] > Min; i>Index} Min > y(t);
```

MinVar je symbol pro konstantu určenou jako minimum pro proměnné daného typu - např. -32768 pro FIXP vstup. Pole Data[1..N] tvoří cyklický zásobník, aktuální místo pro zápis určuje stavová proměnná Index. Funkce MMI je identická až na nahrazení MinVar konstantou MaxVar (32767 pro FIXP vstup) a obrácení smyslu podmínky ... < Max.

Z uvedeného programu je zřejmé, že v režimu studeného startu je celý zásobník jednorázově naplněn aktuální hodnotou vstupu u(t), která tvoří také počáteční hodnotu vystupujícího extrému.

## 4.19 TDL - dopravní zpoždění

#### Určení funkcí a jejich charakteristika

Funkce TDL tvoří pro posloupnost vstupů typu FIXP|FLTP zpožďovací linku s pevně zadaným počtem paměťových buněk. Výstup je tedy "dopravně" zpožděn vůči vstupu:

$$u(t-\text{Delay}) > y(t)$$

přičemž Delay je určeno při překladači jako  $N \cdot T_v$ , kde  $T_v$  je perioda zpracování dané sítě.

Funkce TDL může být použita např. pro konstrukci Smithova prediktoru nebo při vyhodnocování diferencí časových řad s krokem, který je větší než perioda zpracování sítě.

#### Volání funkce TDL

```
U, N > TDL > Y
```

#### Parametry funkce

Typ parametru U určuje typ výstupu Y alternativně jako FIXP nebo FLTP. Druhý vstupní parametr N musí být konstanta typu FIXP - určuje délku cyklického zásobníku, který překladač vyhradí v oblasti statických proměnných (stavových vektorů dynamických funkcí).

#### Zpracování dat ve funkci TDL

```
IF (Studený_start) FOR (1>i; i<=N; i+1>i) u(t) > Data[i];  
Data[Index] > y(y);  
u(t) > Data[Index];  
Index + 1 > Index;  
IF (Index>>N) 1 > Index;
```

## 4.20 SQT - odmocnina

#### Určení funkce a její charakteristika

Funkce SQT vypočítává druhou odmocninu z hodnot vstupního signálu.

#### Volání funkce SQT

```
U > SQT > Y
```

#### Parametry funkce

Vstupní parametr U i výstupní Y je typu FLTP, nebo je na tento typ automaticky konvertován.

#### Zpracování dat ve funkci SQT

Pro záporné vstupy  $u(t) \ll 0$  dává SQT  $0 > y(t)$ , pro nezáporné vstupy vypočítává iteračně přibližnou hodnotu druhé odmocniny.

## 4.21 ABS - absolutní hodnota

#### Určení funkce a její charakteristika

Funkce ABS určuje absolutní hodnotu vstupní proměnné typu FIXP nebo FLTP.

#### Volání funkce ABS

```
U > ABS > Y
```

#### Parametry funkce

Typ parametru U určuje typ výstupu Y alternativně jako FIXP nebo FLTP.

## 4.22 IPR - vstupní normalizace analogových signálů

#### Určení funkce a její charakteristika

Funkce IPR je určena pro prvotní zpracování analogových vstupů získávaných z A/D převodníku. Funkce provádí pouze normalizaci těchto signálů do formátu proměnné typu FIXP. Vstupem funkce IPR mohou být výhradně proměnné ze vstupního pole IL nebo IR; v některých případech zpracovává IPR dvojici po sobě následujících proměnných BYTE, parametrem je však vždy jediná proměnná. Dalším vstupním parametrem je konstanta "Typ\_IPR", která určuje typ normalizace v IPR.

## Volání funkce IPR

`Input, Typ_IPR > IPR > UnifI`

## Parametry funkce

Input - proměnná z pole IL nebo IR

Typ\_IPR - některá z konstant {0,1,2,3,4,5}

UnifI - proměnná FIXP (statická nebo lokální automatická)

## Zpracování dat ve funkci IPR

Unifikaci signálů popíšeme odděleně pro různé "Typ\_IPR":

Typ_IPR	komentář	rozsah vstupu	max(100%)	zpracování
0	2 byte bez změny	<0, 32767>		Input > UnifI
1	1 byte unipolární	<0, 255>	250	Input * 32 > UnifI
2	2 byte unipolární	<0, 4095>	4000	Input * 2 > UnifI
3	1 byte bipolární	<-128, 127>	125	Input * 64 > UnifI
4	2 byte bipolární	<-2048, 2047>	2000	Input * 4 > UnifI
5	2 byte 4 - 20mA	<0, 4095>	4000	(Input*5-4000)/2 > UnifI

Výstupem je ve všech případech proměnná FIXP, u které jmenovitěmu maximumu na vstupu odpovídá hodnota 8000 reprezentovaná standardně jako 100.0%.

## 4.23 OPR - výstupní normalizace analogových signálů

### Určení funkce a její charakteristika

Funkce OPR je určena pro normalizování signálů předávaných k vyslání ze stanice do výstupního pole OL nebo OR a jejich případné saturaci. Vstupem funkce mohou být proměnné typu FIXP, výstupem je proměnná typu BYTE. Dalším vstupním parametrem je konstanta "Typ\_OPR", která určuje typ normalizace.

### Volání funkce OPR

`Outp, Typ_OPR > OPR > UnifO`

### Parametry funkce

Outp - proměnná FIXP nebo BYTE (statická nebo lokální automatická)

Typ\_OPR - konstanta 0 nebo 1

UnifO - proměnná z pole OL nebo OR typu BYTE

### Zpracování dat ve funkci OPR

Typ vstupu Outp	Typ_OPR	maximum (100%)	zpracování vstupu Outp
FIXP	0	255	Outp, 0, 255 > SAT > UnifO
FIXP	1	8000	Outp/32, 0, 250 > SAT > UnifO

Pro Typ\_OPR = 1 se navíc provádí při saturaci výstupu i saturace vstupu! Toto je výjimečná vlastnost standardní funkce - přesto, že vstupní proměnná Outp není zapsána v seznamu výstupních proměnných, ve zvláštním případě saturace výstupu se příslušná hodnota zapíše i do této proměnné. Tato vlastnost se prakticky uplatní jen výjimečně u astatických regulátorů, které nezajistili dostatečně zúžený rozsah výstupu. Saturace v OPR zabrání možnosti neadekvátního nárůstu výstupu S-členů nebo jejich pokles do záporných hodnot - v obou případech klesá kvalita regulačních pochodů.

## 4.24 PIM - pulzní intervalový modulátor

### Určení funkce a její charakteristika

Standardní funkce PIM převádí úroveň analogového vstupního signálu (amplitudově modulovaný signál) na střední hodnotu doby sepnutí jednoho z dvojice výstupních dvouhodnotových signálů. Funkce je určena především pro ovládání servomotorů s konstantní rychlostí (bez polohové zpětné vazby), kdy každý ze dvou logických výstupů odpovídá řízení motoru v jednom směru otáčení a vstup funkce PIM je vlastně požadavkem střední rychlosti pohybu příslušného akčního orgánu. Jiným příkladem použití je řízení středního topného resp. chladicího příkonu, nebo ovládání solenoidových ventilů při míšení tekutin různé teploty nebo chemického složení.

Označíme:

$U$  - amplitudu vstupního analogového signálu

$LS1$  - hodnotu prvního dvouhodnotového výstupního signálu

$LS2$  - hodnotu druhého dvouhodnotového výstupního signálu

$T_v$  - periodu zpracování dat ve funkci PIM

Za předpokladu buzení konstantním signálem  $U$ :

$T_i$  - periodu spínání (nahazování log "1") signálu  $LS_i$

$T_{Ai}$  - dobu sepnutí, tj. šířku výstupního impulsu pro  $LS_i$  (doba rozepnutí, tj. doba trvání mezery mezi impulsy je dána rozdílem  $T_i - T_{Ai} = T_{Pi}$ )

Hodnoty  $T_v$ ,  $T_i$  a  $T_{Ai}$  mají rozměr [s]. Časové údaje  $T_i$  a  $T_{Ai}$  mohou nabývat jen hodnot celočíselných násobků periody  $T_v$ .

Funkce PIM generuje posloupnosti hodnot  $LS1$  a  $LS2$  v závislosti na  $U$  v zásadě podle následujících pravidel:

pro  $U \geq 0$  :  $T_{A1}/T_1 = U/8000$ ,  $T_{A2} = 0$  (LS2 je trvale nulový)

pro  $U < 0$  :  $T_{A2}/T_2 = -U/8000$ ,  $T_{A1} = 0$  (LS1 je trvale nulový)

přičemž parametry  $T_{Ai}$ ,  $T_i$  - tzv. parametry impulzování - určuje algoritmus funkce PIM tak, aby byly splněny podmínky:

$$T_{Ai} \geq T_{A\text{Min}}$$

$$T_i \geq T_{\text{Min}}$$

kde  $T_{A\text{Min}}$  a  $T_{\text{Min}}$  jsou zadané hodnoty parametrů konkrétního volání funkce PIM s významem:

$T_{A\text{Min}}$  - minimální délka výstupního impulsu

$T_{\text{Min}}$  - minimální perioda spínání výstupního signálu

Pro tyto dva parametry musí platit:  $T_{A\text{Min}} \leq T_{\text{Min}}$

Exekutiva funkce PIM určuje  $T_{Ai}$  a  $T_i$  ve dvou režimech:

pro  $(U > \text{ABS}) \gg (8000 * T_{A\text{Min}} / T_{\text{Min}})$  jako:

$$(U > \text{ABS}) * T_{\text{Min}} / 8000 > T_{Ai}; \quad T_{\text{Min}} > T_i;$$

pro  $(U > \text{ABS}) \leq (8000 * T_{A\text{Min}} / T_{\text{Min}})$  jako:

$$T_{A\text{Min}} > T_{Ai}; \quad T_{A\text{Min}} * 8000 / (U > \text{ABS}) > T_i;$$

Jde tedy pro větší absolutní hodnoty vstupu  $U$  o šířkovou modulaci s pevnou periodou minimální povolené velikosti  $T_{\text{Min}}$  a pro menší absolutní hodnoty  $U$  o frekvenční modulaci s pevnou šířkou impulsu minimální povolené velikosti  $T_{A\text{Min}}$ . Pro  $T_{A\text{Min}} = 0$  se PIM chová pro všechny hodnoty  $U$  jako šířkový modulátor, pro  $T_{A\text{Min}} = T_{\text{Min}}$  se PIM chová jako modulátor frekvenční.

Převodník PIM provádí kvantování signálu - více různých hodnot  $U$  vede k impulzování se stejnými parametry  $T_{Ai}$ ,  $T_i$ .

Pro šířkovou modulaci je vstup U kvantován po:

$$\Delta = 8000 * T_v / T_{Min}$$

Pro frekvenční modulaci je vstup kvantován po:

$$\Delta = 8000 * T_v * T_{AMin} / T_{Min} * (T_{Min} + T_v)$$

Absolutní chyba kvantování je nejvýše  $\Delta/2$ .

Volbou parametrů  $T_{AMin}$  a  $T_{Min}$  určujeme tedy mezní hodnotu vstupu U pro přechod od šířkové modulace k modulaci frekvenční a naopak, "jemnost" kvantování v obou režimech a dále také schopnost převodníku přenášet různé frekvence vstupního signálu. Z výše uvedených vztahů vyplývá, že PIM dává pro velké hodnoty U větší chybu kvantování, pro malé hodnoty U je chyba kvantování menší. Maximální relativní chyby kvantování jsou:

$$\text{pro } (U > ABS) \gg (8000 * T_{AMin} / T_{Min}) \quad \Delta_{Rmax} = 4000 * T_v / ((U > ABS) * T_{Min})$$

$$\text{pro } (U > ABS) \leq (8000 * T_{AMin} / T_{Min}) \quad \Delta_{Rmax} = T_v / 2 * (T_{Min} + T_v)$$

Je zřejmé, že s klesajícím  $T_v$  jednoznačně klesá chyba kvantování.  $T_{AMin}$  volíme s ohledem na požadovanou přesnost ovládání u servomotorů např. jako nejmenší šířku impulsu napájení, který vyvolá definované pootočení hřídele na výstupu převodovky.

Obtížnější může být volba  $T_{Min}$ . Velké hodnoty  $T_{Min}$  dávají sice menší chyby kvantování při šířkové modulaci, ale snižují také absolutní velikost mezního vstupu ( $U > ABS$ ) pro přechod k frekvenční modulaci, která dává menší chyby kvantování než modulace frekvenční. Pro  $T_{Min}$  daleko větší než  $T_v$  se může projevit také necitlivost převodníku PIM při průchodu signálu U nulou způsobená tím, že po změně polarity nebo po vynulování signálu U začíná příslušný signál  $LS_i$  vždy mezerou s dobou trvání ( $T_i - T_{Ai}$ ); jestliže během této doby signál U opět změnil polaritu začne alternativní  $LS_i$  opět mezerou ... znamená to, že pravouhlý periodický signál s frekvencí  $f$  a maximální absolutní hodnotou  $u_M$  vyvolá trvale nulové oba výstupy PIM jestliže:

$$1/2f \ll T_i * (1 - u_M / 8000)$$

resp.  $1/2f \ll T_{Min} * (1 - u_M / 8000)$  pro šířkovou modulaci

a  $1/2f \ll T_{AMin} / u_M * (8000 - u_M)$  pro frekvenční modulaci

Menší hodnota  $T_{Min}$  vede k větší frekvenci spínání a tím i k většímu zatěžování a opotřebením akčního orgánu při středně velkých hodnotách U. Při velkých hodnotách  $T_{Min}$  se PIM chová téměř jako čistý šířkový modulátor se zaručenou maximální frekvencí spínání (počet sepnutí za hodinu je  $\leq$  hodnotě  $3600/T_{Min}$ ), skutečný počet sepnutí je možné odhadovat na základě histogramu - odhadu hustoty pravděpodobnosti pro vstup U. Při extrémně malé hodnotě  $T_{Min}$  se PIM chová téměř při všech hodnotách U kromě největších jako frekvenční modulátor.

### Extrémní hodnoty vstupu a jejich zpracování

Nulový vstup  $U=0$  vyvolá vynulování obou výstupů  $LS_1$  a  $LS_2$  a nastavení výchozího "neutrálního" vnitřního stavu modulátoru (vynulování čítače počtu period udávajícího délku trvání posledního impulsu resp. mezery mezi impulsy).

V absolutní hodnotě nejmenší nenulový vstup, který vyvolá generaci výstupního impulsu má velikost 1 a maximální perioda je:

$$T_i = T_{AMin} * 8000$$

Vzhledem k omezení rozsahu vnitřního čítače platí pro periodu výstupních signálů  $LS_i$  omezení:

$$T_i \leq (32767 * T_v, 8000 * T_{AMin} > MIN)$$

Statická charakteristika převodníku je, jak vyplývá z výše uvedeného, schodovitá funkce. Trvalé sepnutí  $LS_i$  dostaneme pro:

$$(U > ABS) \geq (8000 - 4000 * T_v / T_{Min})$$

Z uvedeného dále plyne, že funkce PIM je "dynamická" - každé volání této funkce vytváří objekt s vlastním stavovým vektorem alokovaným mezi statickými proměnnými. Tento vektor zachycuje dvouhodnotové informace o znaménku vstupu U v minulém kroku a o fázi "impuls" nebo "mezera" a zahrnuje dvoubytový čítač aktuální délky impulsu či mezery mezi impulsy. Podobně jako funkce regulátorů a časových členů proto nelze funkci PIM používat v těle cyklů.

## Formát volání funkce PIM

Funkce PIM má tři vstupní parametry U, TAMin, TMin, z nichž poslední dva musí být konstantami typu FIXP. Výstupy typu BOOL se zadávají v pořadí: Výstup pro záporné vstupy, Výstup pro kladné vstupy.

Je tedy formát volání:

```
U, TAMin, TMin > PIM > LS2, LS1
```

## Zpracování dat ve funkci PIM

V každé periodě zpracování dat je vstup omezen do intervalu <-8000,8000> a dále jsou určeny v souladu s výše uvedenými vztahy parametry impulzování TAI a Ti. Znaménko vstupu U určí jako aktivní výstup LS1 nebo LS2. Změna polarit vstupu U nebo jeho nulová hodnota vyvolá vynulování čítače. Pokud v minulé periodě byl některý výstup aktivní (ve stavu "1"), je momentální hodnota inkrementovaného čítače komparována s TAI, v opačném případě s (Ti-TAI), tedy s požadovanou dobou trvání mezery mezi impulsy. Jestliže momentální hodnota U určí délku impulsu, která již byla dosažena nebo překročena, je impuls ukončen a čítač vynulován. Podobně v době trvání mezery mezi impulsy může klesající absolutní hodnota vstupu způsobit postupné prodlužování této mezery a obráceně. Ukončení mezery vyvolá vynulování čítače a start nového impulsu. Je zřejmé, že vliv rychlejších změn vstupu na výstupy je prakticky okamžitý.

## 4.25 REC - zápis do cyklického zásobníku

### Určení funkce a její charakteristika

Standardní funkce REC slouží pro postupný periodický nebo aperiodický sběr vzorků zadané veličiny a jejich ukládání do cyklického zásobníku konstantní délky. Sběr dat lze startovat a zastavovat prostřednictvím vstupního BOOL parametru "Start", sestupná hrana tohoto signálu vyvolává přepis aktuální hodnoty indexu cyklického zásobníku do proměnné tzv. "zmrazeného indexu", která může sloužit ve funkci [CER](#) a reprezentacích [\\$TIGR](#) a [\\$TAB](#) k výběru uložených hodnot počínaje časovým indexem, který je svázán s nějakou událostí návštěvou právě sestupnou hranou signálu Start. Je-li Start == 0 jen při jednom vyvolání konkrétní funkce REC, dojde sice ke zmrazení indexu, ale sběr dat pokračuje bez přerušení při každém dalším vyvolání. Je-li Start == 0 ve více bezprostředně následujících voláních, je sběr dat zastaven až do přechodu tohoto vstupu do Start == 1.

Funkce REC je funkcí dynamickou, a to v tom smyslu, že má podobně jako PID, FIL, CNT, SDR, ... , své stavové proměnné (aktuální index, paměť pro Start(t-1) a vlastní cyklický zásobník). Tyto funkce jsou v jazyce LEDA užívány vlastně jako objekty, které mají svou procedurální část, ale i datovou část, která je obecně uživateli přístupná jen částečně a zprostředkovaně. Na rozdíl od ostatních dynamických funkcí nevzniká pro každé volání REC samostatný objekt, ale všechna volání s totožným parametrem Zdroj - proměnnou jejíž hodnoty jsou přenášeny do cyklického zásobníku - vytvářejí jediný objekt, tzn., že mj. vedou k alokaci jediného cyklického zásobníku. Navíc i různá volání funkce CER a reprezentací \$TIGR a \$TAB mohou zavádět prostřednictvím společného identifikátoru proměnné Zdroj vazbu na jediný objekt REC. Lze tedy v různých místech programu ukládat do jediného zásobníku momentální hodnoty proměnné Zdroj a na jiných místech se lze do zásobníku odkazovat a hodnoty z něj lze vybírat a různým způsobem zpracovávat.

### Formát volání funkce REC

Formát volání je pevný, se třemi vstupními parametry a bez výstupu:

```
Start, Zdroj, Počet_vzorků > REC
```

### Parametry funkce

Start - BOOL proměnná pro řízení záznamu - náběžná hrana startuje záznam, Start==0 deaktivuje REC;

Zdroj - Proměnná typu BYTE | FIXP | FLTP, ze které je hodnota při splnění podmínky (Start(t)|Start(t-1)) při každém vyvolání REC se shodným identifikátorem Zdroj přenesena do cyklického zásobníku;

Počet\_vzorků - Celková kapacita zásobníku - konstanta FIXP (>>0).

## 4.26 CER - čtení dat z cyklického zásobníku

### Určení funkce a její charakteristika

Funkce CER je svým způsobem "inverzní funkcí" vůči funkci REC - může sloužit jako výchozí krok při zpracovávání posloupností historických vzorků vytvářených funkcí (popř.funkcemi) REC. Standardní funkce CER zajišťuje vyvolávání skupin hodnot z určeného cyklického zásobníku REC a jejich přenášení do jednorozměrného pole. První hodnota přenášená do výstupního pole může být čtena alternativně podle aktuálního nebo zmrazeného indexu zásobníku REC, s posunem směrem ke starším vzorkům určeným okamžitou hodnotou vstupního výrazu tvořícího první parametr funkce CER. Dále je čten a do výstupního pole přenášen každý N-tý vzorek ze zásobníku REC, přičemž N je konstantní vstupní parametr "Krok". Celkový počet hodnot vybíraných ze zásobníku REC je určen okamžitým rozměrem výstupní matice (vektoru).

### Formát volání funkce CER

Formát volání je pevný, se čtyřmi vstupními parametry a jedním výstupním parametrem:

```
Posun, Index_aktuální/zmrazený, Krok, Zdroj > CER > Matice
```

### Parametry funkce

Posun - Výraz typu FIXP určující "stáří" prvního vzorku čteného ze zásobníku REC.

Index\_aktuální/zmrazený - Řetězcová konstanta "C"|"F" příznaku aktuální|zmrazený index REC.

Krok - Konstanta typu BYTE určující krok výběru ze zásobníku REC.

Zdroj - Proměnná typu BYTE|FIXP|FLTP určující konkrétní záznam REC - tato proměnná musí být použita alespoň v jednom volání funkce REC na místě jejího parametru Zdroj.

Matice - Jednorozměrná matice určená k zápisu hodnot čtených ze zásobníku REC - tato matice je obecně částí pole implicitních proměnných nebo deklarovaného pole.

Při čtení posloupnosti vzorků ze zásobníku REC není kontrolována jeho délka ve vztahu k počtu prvků výstupní matice CER; to prakticky znamená, že určitý vzorek může být při jednom volání CER čten a zapsán do výstupní matice vícenásobně.

## 4.27 OUTF - výstup do znakových souborů a na tiskárnu

### Určení funkce a její charakteristika

Standardní funkce OUTF zajišťuje generaci znakového řetězce a jeho zápis do diskového souboru, popřípadě výpis na tiskárnu. Řetězec je generován na základě momentálních hodnot proměnných tvořících vstupní parametry funkce OUTF, prostřednictvím reprezentací, které jsou dalšími parametry OUTF (zajistí převody hodnot proměnných na údaje ve fyzikálních jednotkách včetně symbolu fyzikální jednotky) a dále může být vystupující řetězec doplněn ve funkci komentářů řetězci (proměnnými nebo konstantami), které jsou rovněž vstupními parametry funkce. Výstupní řetězec je pouze posloupností ASII znaků - neobsahuje ani údaj délky ani koncový znak. Výstupním parametrem funkce OUTF je řetězová konstanta, která popisuje "cestu" (posloupnost adresářů) a jméno výstupního souboru. Přípona je generována v okamžiku volání funkce OUTF automaticky podle hodnoty systémové proměnné YDAY, tedy jako pořadové číslo dne v roce (čísla 001 až 366). Výstup do tohoto souboru proběhne za předpokladu, že soubor daného jména již existuje, pokud tomu tak není, je soubor automaticky založen.

Z uvedených charakteristik je zřejmé, že funkce OUTF je určena k vytváření deníku různých událostí - pro výstražná a alarmová hlášení, periodické výstupy bilančního charakteru nebo pro vytváření deníku zásahů dispečera, deníku servisních úkonů a pod.. Pro pravidelné výpisy bude volání OUTF podmíněno např. změnou hodinového údaje (HOURL>DIF), pro poruchové výpisy vyhodnocením různých boolských výrazů. Funkce OUTF je použitelná výhradně v operátorské stanici (PC vybavené diskovou resp. disketovou jednotkou).

TOLEDA zajišťuje zakládání dosud neexistujících souborů daného jména a s příponou určenou právě platným pořadovým číslem dne v roce. Je zřejmé, že nejméně jedenkrát za rok je vhodné provést revizi obsahu diskových

souborů a neaktuální soubory vymazat, respektive vyprázdnit. Do existujících diskových souborů funkce OUTF připsuje.

### Volání funkce OUTF

Funkce OUTF má proměnný počet parametrů - maximálně 64. Seznam vstupních parametrů je tvořen dvojicemi:

Výraz, Id\_R > OUTF > „Jmeno\_souboru“

### Parametry funkce

Výraz - výraz typu BYTE|FIXP|FLTP nebo řetězcová proměnná či konstanta

Id\_R - identifikátor reprezentace s výstupním typem CHAR.

Je-li Výraz řetězcem, Id\_R se nezadáva, tzn., že se do seznamu vstupních parametrů zapíše jen povinná čárka.

Jediným výstupním parametrem funkce OUTF je řetězcová konstanta udávající jméno výstupního diskového souboru nebo standardní jméno výstupního zařízení používané v MS DOS - např. "lpt1" určuje jako výstupní periférii tiskárnu připojenou k prvnímu paralelnímu portu.

### Zpracování dat ve funkci OUTF

Každá dvojice parametrů Výraz, Id\_R určuje adresu hodnoty daného typu a reprezentaci \$Id\_R, přes kterou je tato hodnota převedena na řetězec. Všechny řetězce vzniklé zpracováním zadaných reprezentací jsou sjednoceny s přímo vstupujícími řetězci (bez přídatných mezer nebo jiných znaků), a tím je vytvořen jediný výsledný řetězec, který je okamžitě - v souladu s výstupním parametrem - zařazen do výstupního proudu dat.

## 4.28 OUTFS - generace znakového pole číselných údajů

### Určení funkce a její charakteristika

Standardní funkce OUTFS zajišťuje generaci znakového pole určeného pro vyslání z technologické řídicí stanice do terminálu operátora. Výstupní znakové pole je generováno na základě momentálních hodnot proměnných tvořících vstupní parametry funkce OUTFS, prostřednictvím reprezentací, které jsou dalšími parametry OUTFS (zajistí převody lineárních hodnot na údaje ve fyzikálních jednotkách včetně symbolu fyzikální jednotky) a dále může být vystupující pole znaků doplňováno komentáři, které jsou rovněž - ve formě přímo vstupujících řetězců - vstupními parametry funkce. Výstupní pole je pouze posloupností ASCII znaků - neobsahuje ani údaj délky ani koncový znak.

### Vyvolání funkce OUTFS

Funkce OUTFS má proměnný počet parametrů - maximálně 64. Seznam vstupních parametrů je tvořen dvojicemi:

Výraz, Id\_R > > OUTFS > Pole\_znaku

### Parametry funkce

Výraz - výraz typu BYTE|FIXP|FLTP nebo řetězcová proměnná či konstanta

Id\_R - identifikátor reprezentace s výstupním typem CHAR.

Je-li Výraz řetězcem, Id\_R se nezadáva, tzn., že se do seznamu vstupních parametrů zapíše jen povinná čárka.

Jediným výstupním parametrem funkce OUTFS je pole typu CHAR.

### Zpracování dat ve funkci OUTFS

Každá dvojice parametrů Výraz, Id\_R určuje adresu hodnoty daného typu a reprezentaci \$Id\_R, přes kterou je tato hodnota převedena na řetězec. Všechny řetězce vzniklé zpracováním zadaných reprezentací jsou sjednoceny s přímo vstupujícími řetězci (bez přídatných mezer nebo jiných znaků), a tím je vytvořeno výsledné znakové pole.

## 4.29 OUTR - inicializace vyslání znakové zprávy

### Určení funkce a její charakteristika

Vyvoláním funkce OUTR dojde k předání textové zprávy umístěné v poli OR do vyrovnávací paměti zprávy určené k vyslání sériovou linkou do nadřazené operátorské stanice - typicky do přenosného terminálu s LCD zobrazovačem - a k inicializaci vyslání této zprávy.

### Vyvolání funkce OUTR

Funkce OUTR nemá žádný vstupní parametr - jediný výstupní parametr, kterým je číselná konstanta 0 nebo 1, určuje, zda se jedná o vyslání textové zprávy z první poloviny pole OR (OR00 až ORFF) na sériovou linku s logickým číslem 0, nebo o vyslání zprávy z druhé poloviny pole OR (OR100 až OR1FF) na linku s logickým číslem 1.

## 4.30 ATN - převod pole znaků na číselnou hodnotu

### Určení funkce a její charakteristika

Funkce ATN provádí v souladu se zadanou reprezentací se vstupním typem CHAR převod skupiny nejvýše N ASCII znaků na číselnou hodnotu a její uložení do výstupní proměnné. Série vstupních znaků je přitom podrobena syntaktické kontrole, jejíž výsledek určuje hodnotu výstupního parametru typu BOOL. Vstupní čísla ve znakové reprezentaci musí vyhovovat syntaktickým pravidlům číselných konstant jazyka LEDA. Funkce ATN může být využita např. při zpracovávání textových zpráv z terminálu operátora nebo jiných zařízení vysílajících textové zprávy.

### Vyvolání funkce ATN

Formát vyvolání funkce ATN je následující:

```
PoleZnaků, Reprezentace, N > ATN > Err, Výstup
```

### Parametry funkce

Reprezentace - identifikátor reprezentace se vstupním typem CHAR a výstupním typem BYTE|FIXP|FLTP

N - konstanta typu BYTE udávající maximální počet zpracovávaných znaků z pole PoleZnaků

Err - proměnná typu BOOL - hodnota 1 indikuje chybné zadání čísla

Výstup - proměnná typu BYTE|FIXP|FLTP - výsledek převodu

Proměnnou Err funkce ATN v případě chyby nastavuje, ale při bezchybném převodu nenuluje! Proměnná Err tak může sloužit např. pro několik volání funkce ATN jako globální příznak chybného zpracování vstupů a může být explicitně programově shozen např. až při pokusu o nový převod zprávy.

## 4.31 SRG - posuvný registr

### Určení funkce a její charakteristika

Funkce SRG může být používána pro realizaci číslicových korekčních členů při zpětnovazebním řízení (regulaci), realizaci generátorů, filtrů a prediktorů (zpětnovazebních nebo s konečnou pamětí) a obecně ve všech situacích, kdy je potřebné pracovat s posloupnostmi vzorků různých analogových signálů - mj. např. pro zobrazování časových průběhů veličin formou bodových grafů (\$TIGR).

### Vyvolání funkce SRG

Formát volání je pevný:

```
Vstup > SRG > Pole;
```

### Parametry funkce

Vstup - výraz typu BYTE | FIXP | FLTP

Pole - jednodimenzionální pole typu BYTE | FIXP | FLTP

### Zpracování dat ve funkci SRG

```
FOR( MaxIndexPole > i; i>=MinIndexPole; i-1 > i) Pole[i-1] > Pole[i];  
Vstup > Pole[MinIndexPole];
```

Jsou tedy při každém vyvolání přepsány všechny starší vzorky signálu Vstup na místa vzorků o jeden krok (většinou o pevnou periodu zpracování dané sítě) starších – nejstarší vzorek je přepsán druhým nejstarším, ... Aktuální hodnota proměnné Vstup potom přepíše prvek pole Pole s nejnižším (obvykle nulovým) indexem. Po dostatečném počtu vyvolání SRG ( $\text{MaxIndexPole} - \text{MinIndexPole} + 1$ ) je výsledkem aktualizovaná posloupnost vzorků signálu Vstup uložených v Pole tak, že vyšším indexům odpovídá větší stáří vzorků.

Typ proměnné Vstup nemusí souhlasit s typem pole Pole – SRG zajišťuje konverzi typu, nikoli však saturace při převodu z „delšího“ na „kratší“ typ.

### Poznámka

Použití standardní funkce SRG je z hlediska časového zatížení výhodné obvykle jen pro kratší záznamy historie signálů (max. desítek vzorků); pro rozsáhlejší záznamy, z nichž je nutné získávat pro další zpracování vždy jen kratší úseky, je vhodnější volit metodu nevyžadující při každé aktualizaci přepis (posun) všech starších vzorků, tj. cyklický zápis s indexem  $i$  ukazujícím na aktuální vzorek, inkrementovaným těsně před zápisem do pole a pro  $i > \text{MaxIndexPole}$  nastavovaným na  $\text{MinIndexPole}$ .

## 4.32 CPS - sdružování skupin BOOL proměnných

### Určení funkce a její charakteristika

Funkce CPS plní výstupní proměnnou typu BYTE nebo FIXP hodnotou vytvořenou jako binární ekvivalent hodnot skupiny vstupů typu BOOL. Funkce umožňuje provádět kompresi dvouhodnotových signálů pro zefektivnění následného zpracování, přenosu nebo archivace. Např. pro klasifikace různých událostí pomocí [SWI](#) je potřebné sloučit v jediné proměnné několik BOOL signálů získaných z různých vstupních karet a pod. Rovněž např. pro historické záznamy realizované pomocí standardní funkce [REC](#) je sdružování skupiny BOOL signálů užitečné. Generování indexů na základě výrazů typu BOOL se často uplatní při předzpracování dat pro vizualizaci pomocí standardních reprezentací [\\$GTEXT](#).

### Vyvolání funkce CPS

Počet vstupních parametrů je proměnný v rozsahu 1 až 16, výstupní parametr je jediný:

```
Bn, ..., B4, B3, B2, B1, B0 > CPS > Vystup
```

Počet BOOL vstupů větší než 7 vyžaduje Vystup typu FIXP, pro menší počet vstupů postačuje Vystup typu BYTE.

### Zpracování dat ve funkci CPS

Hodnoty jednotlivých BOOL vstupů  $B_n .. B_0$  určují jednotlivé bity výstupní proměnné Vystup, a to tak, že vstup  $B_n$  určuje  $n$ -tý bit výstupu a vstup  $B_0$  určuje 0-tý bit výstupu. Poskytuje tedy funkce CPS stejné výsledky jako vyhodnocení výrazu:

$$B_n * 2^n + \dots + B_4 * 2^4 + B_3 * 2^3 + B_2 * 2^2 + B_1 * 2 + B_0$$

## 4.33 DCPS - rozklad proměnných na skupiny BOOL

### Určení funkce a její charakteristika

Funkce DCPS je "inverzní" funkcí vůči funkci CPS - DCPS umožňuje "rozeslat" jednotlivé bity proměnné typu FIXP až na 16 proměnných typu BOOL. Funkci lze používat pro "dekompresi" dat přijatých ze vzdálených stanic nebo archivovaných dat a pro zefektivnění některých operací např. při vysílání výstupů sekvenčních automatů (automat

může nastavovat pomocné slovo nebo byte a na závěr zpracování jsou pomocí DCPS rozeslány BOOL výstupy na definitivní adresy, anebo je v každé větvi automatu volána funkce DCPS se vstupem tvořeným číselnou konstantou zapsanou v reprezentaci `$BB` nebo `$B`).

### Vyvolání funkce DCPS

Vstupní parametr je jediný, počet výstupních parametrů je proměnný v rozsahu 1 až 16:

```
Vstup > DCPS > Bn, ..., B4, B3, B2, B1, B0
```

Vstup typu BYTE je automaticky konvertován na typ FIXP.

### Zpracování dat ve funkci DCPS

Hodnoty jednotlivých BOOL výstupů Bn .. B0 jsou určovány jednotlivými bity vstupní proměnné Vstup, a to tak, že výstup Bn je určen n-tým bitem vstupu a výstup B0 je určen 0-tým bitem vstupu.

## 4.34 BEEP - tónový výstup

### Určení funkce a její charakteristika

Funkce BEEP je určena pro realizaci jednoduchého akustického výstupu v operátorské stanici - je použitelná pouze v MASTER sítích aplikačního programu. Funkce má jediný vstupní parametr typu BOOL, jehož náběžná hrana určuje okamžik startu tónového výstupu tvořeného dvěma střídajícími se tóny různé - pevně určené - frekvence. Ukončení tohoto výstupu je možné výhradně stisknutím klávesy mezerníku na standardní klávesnici osobního počítače.

### Vyvolání funkce BEEP

Formát vyvolání funkce BEEP je následující:

```
Start > BEEP;
```

### Parametry funkce

Start - výraz typu BOOL.

## 4.35 AUDIO - obecný tónový výstup

### Určení funkce a její charakteristika

Funkce AUDIO umožňuje řídit akustický výstup na osobním počítači - lze ji vyvolávat pouze v MASTER sítích. První vstupní parametr typu BOOL určuje svojí náběžnou hranou okamžik zahájení interpretace posloupnosti parametrů tónů a pomlka, které byly před vyvoláním funkce uloženy do pole typu FIXP tvořícího její druhý vstupní parametr. Koncová značka v tomto poli vyvolává nový start interpretace od prvního prvku. Ukončení interpretace zadané "melodie" je možné výhradně stisknutím klávesy mezerníku na standardní klávesnici osobního počítače.

### Vyvolání funkce AUDIO

Formát vyvolání funkce AUDIO je následující:

```
Start, Melody > AUDIO;
```

### Parametry funkce

Start - výraz typu BOOL určující zahájení interpretace melodie,

Melody - pole typu FIXP tvořené posloupností dvojic parametrů: Frekvence, Délka

Frekvence se zadává v [Hz],

Délka se zadává jako počet základních časových prodlev po 1/18 [s], tj. s kvantováním cca po 55.5 [ms].

Dvojice Frekvence-Délka začínají od 1.prvku pole Melody; 0-tý prvek je rezervován pro momentální hodnotu indexu tónu.

Pomlka se zadává jako tón s nulovou frekvencí, koncovou značku tvoří libovolné záporné číslo.

Výšku tónů, jejich délku a délku pomlky lze "ladit" i v průběhu interpretace, přepsáním některé frekvence lze manuálním zásahem nebo i algoritmicky zkracovat cyklus nebo "zacyklením" na první pomlce výstup fakticky "vypnout" i když nebyla interpretace melodie ukončena zásahem z klávesnice. Index aktuálního tónu lze rovněž modifikovat. Současně nelze samozřejmě generovat více než jednu posloupnost tónů.

## 4.36 WRITEC - zápis záznamu do archivního diskového souboru

### Určení funkce a její charakteristika

Funkce WRITEC připsuje do binárního diskového souboru jeden záznam tvořený skupinou momentálních hodnot N vstupních parametrů typu BYTE, FIXP nebo FLTP a do stejného souboru připsuje také údaj časové značky určené v okamžiku vyvolání na základě momentálních hodnot systémového data SDATE (YEAR, MONT, DAY) a systémového času STIME (HOUR, HMIN, MSEC). Jméno výstupního souboru včetně označení disku a posloupnosti adresářů tvoří jako řetězcová konstanta výstupní parametr funkce WRITEC. Maximální počet záznamů, které je možné zapsat do souboru, je určen posledním vstupním parametrem typu FIXP - každý následující záznam přepisuje vždy nejstarší záznam.

Funkce WRITEC postupně plní binární diskový soubor jako archivní pro skupinu určených proměnných. Skupinu záznamů pořízenou vícenásobným voláním funkce WRITEC a určenou intervalem časových značek umožňuje ze souboru přečíst a zapsat do výstupních polí funkce READC.

Archivace v binárních cyklických souborech může být periodická nebo řízená událostmi - může jít např. o archivaci výstražných a alarmových stavů nebo o vytváření deníku zásahu operátora.

Archivní soubory mohou sloužit mj.také pro výměnu dat mezi TOLEDA a jinými programy (např. komunikačními programy instalovanými jako rezidentní programy).

### Vyvolání funkce WRITEC

Funkce má proměnný počet N+1 vstupních parametrů a jediný výstupní parametr:

```
Vstup1, Vstup2, ..., VstupN, MaxPocetZaznamu > WRITEC > "JmenoSouboru";
```

### Parametry funkce

Vstup1, ..., VstupN - archivované signály, jejichž vzorky tvoří záznam zapisovaný do souboru - jde o jednoduché proměnné typu BYTE nebo FIXP nebo FLTP. Celková délka záznamu tvořená součtem délek všech těchto proměnných nesmí překročit 240 byte! Pro větší objem archivovaných dat (s toutéž periodou) je nutné použít několika archivních souborů a příslušných volání WRITEC.

MaxPocetZaznamu - konstanta typu FIXP určující maximální počet záznamů uložitelných do souboru,

"JmenoSouboru" - řetězcová konstanta určující disk, adresář a jméno binárního diskového souboru.

### Zpracování dat ve funkci WRITEC

Funkce WRITEC otevře existující soubor daného jména "JmenoSouboru", anebo tento soubor vytvoří. Údaj okamžitého systémového data a času z SDATE a STIME je převeden na jediný čtyřbytový údaj "počet sekund od začátku r.1993" a zapsán do oblasti časových značek daného souboru (s těmito časovými značkami mohou pracovat funkce READC, AKTC a LASTTC se shodným parametrem "JmenoSouboru").

Z okamžitých hodnot vstupů Vstup1 až VstupN je vytvořen záznam (bez jakýchkoli přídavných údajů), který je zapsán na místo nejstaršího záznamu. Na závěr WRITEC daný diskový soubor uzavře.

### Důležité poznámky

Má-li být v programu volána funkce READC nebo AKTC či LASTTC, musí být uvedena také funkce WRITEC se stejným jménem archivního souboru! Parametry volání WRITEC slouží překladači k určení údajů poskytovaných všem

ostatním funkcím operujícím na témže souboru - jedná se o strukturu jednotlivých záznamů a jejich celkový počet. Pokud má být archivní soubor plněn výhradně funkcí [AKTC](#), je potřeba volání WRITEC uvést jako podmíněné s trvale nesplněnou podmínkou.

Strukturu souboru nelze při úpravách programu modifikovat, jestliže není zajištěno spuštění na počítači, na jehož pevném disku neexistuje dříve z TOLEDA plněný soubor se jménem "JmenoSouboru"! Starší archivní soubory s odlišnou strukturou nebo jinou maximální délkou musí být přejmenovány, popř. zrušeny.

Struktura cyklického binárního diskového souboru je následující

Počet byte	Obsah
3	Datum posledního záznamu: rok, měsíc, den
3	Čas posledního záznamu : hodina, minuta, sekunda
2	Index posledního záznamu
4	Sekunda záznamu 0
4	Sekunda záznamu 1
	.....
DelkZazn	Zaznam 0
DelkZazn	Zaznam 1
	.....

Přičemž DelkZazn je celková délka záznamu určená součtem délek všech proměnných Vstup1 .. VstupN podle jejich typů.

## 4.37 READC - čtení záznamů daného čas.intervalu z archiv.souboru

### Určení funkce a její charakteristika

Funkce READC čte z binárního diskového souboru (naplněného voláním funkce [WRITEC](#) popř. [AKTC](#)) skupinu záznamů určenou intervalem časových údajů s významem "Po čas.značce" a "Do čas.značky" tvořících vstupní parametry READC a zapisuje údaje data zápisu, času zápisu a vybrané archivované vzorky postupně do výstupních polí tvořících první skupinu výstupních parametrů funkce READC. Maximální počet záznamů čtených ze souboru je omezen počtem prvků výstupních polí. Funkce READC dále poskytuje údaj skutečného počtu záznamů zapsaných do výstupních polí a příznak pravděpodobné ztráty archivovaných dat v případě, že požadovaný časový interval začíná před okamžikem zápisu nejstaršího záznamu.

Pomocí funkce READC lze číst vybraná archivovaná data z diskového souboru a plnit jimi pole, která mohou být dále zpracovávána např. pro účely optimalizace parametrů zpětnovazebního řídicího systému, pro bilanční výpočty, vyhodnocování mimolimitních stavů, analýzu zásahů operátora a pod. Posloupnosti vzorků jednotlivých archivovaných signálů mohou být také bez předchozího zpracování v exekutivě systému vizualizovány s užitím standardních reprezentací typu \$TAB nebo \$TIGR. Rozsáhlejší archivní soubory lze pomocí READC číst po částech zvolené maximální velikosti (počet prvků výst.polí) v cyklu, nebo ve více po sobě následujících periodách zpracování dat. Interval časových značek může řídit také operátor přímým zadáním časových mezí popř. volbou "dříve" | "později".

Funkce READC se nemůže v programu vyskytovat bez příslušného volání (byť formálního) funkce WRITEC se shodným jménem archivního souboru (podrobnosti viz popis funkce [WRITEC](#)).

### Vyvolání funkce READC

Funkce má proměnný počet M+5 vstupních parametrů a stejný počet výstupních parametrů:

```
DatumOd, CasOd, DatumDo, CasDo, Promenna1, Promenna2, ..., PromennaM,
    "JmenoSouboru" > READC > PoleDatum, PoleCas,
Pole1, Pole2, ..., PoleM, PocetUdalosti, Preplneni, Ztrata;
```

### Parametry funkce

DatumOd - pole typu BYTE ARRAY[#3] (rok-1900,měsíc,den)

CasOd - pole typu BYTE ARRAY[#3] (hodina,minuta,sekunda)

DatumDo - pole typu BYTE ARRAY[#3] (rok-1900,měsíc,den)

CasDo - pole typu BYTE ARRAY[#3] (hodina,minuta,sekunda)

Promenna1,Promenna2,...,PromennaM - proměnné, které jsou podmnožinou množiny proměnných tvořících vstupní parametry ve volání funkce WRITEC zapisující do stejného souboru

"JmenoSouboru" - řetězcová konstanta určující jednak soubor, ze kterého má být čteno, jednak svazuje volání READC s konkrétním voláním funkce WRITEC, v němž je určen rozsah souboru (max.počet záznamů) a úplná množina vzorkovaných proměnných

PoleDatum - pole typu BYTE ARRAY[#M][3]

PoleCas - pole typu BYTE ARRAY[#M][3]

Pole1,...,PoleM - jednorozměrná pole typu odpovídajícího typům archivovaných proměnných Promenna1,...,PromennaM

PocetUdalosti - proměnná typu FIXP jejíž hodnota udává skutečný počet vyhledaných záznamů a vzorků zapsaných do výstupních polí

Preplneni - BOOL proměnná - příznak vyčerpání kapacity výst.polí

Ztrata - BOOL proměnná - příznak pravděpodobné ztráty dat (čas. interval sahá dále do minulosti, než udává časová značka nejstaršího záznamu)

### Zpracování dat ve funkci READC

Funkce READC otevře existující soubor daného jména "JmenoSouboru", anebo ukončí okamžitě svou činnost v případě, že soubor daného jména nemůže otevřít (neexistuje cesta nebo soubor). Hodnoty parametrů DatumOd a CasOd jsou převedeny na údaj "Po čas.značce", hodnoty parametrů DatumDo a CasDo jsou převedeny na údaj "Do čas.značky". Je nalezen záznam s časovou značkou nejbližší vyšší než udává "Po čas.značce". Počínaje tímto záznamem jsou čteny novější záznamy a vybírány z nich vzorky patřící proměnným Promenna1 .. PromennaM a tyto vzorky jsou zapisovány do výstupních polí počínaje jejich nejnižším indexem. Do polí PoleDatum a PoleCas jsou zároveň zapisovány údaje data a času provedení záznamu získávané převodem časových značek ze souboru, a to ve formátu YEAR, MONT, DAY a HOUR, HMIN, MSEC. Po vypsání posledního požadovaného záznamu s časovou značkou "Do čas.značky" nebo po provedení zápisu do prvků výstupních polí s nejvyšším povoleným indexem, popř. po zpracování nejnovějšího záznamu v souboru je čtení a zpracování záznamů ukončeno a výsledky hledání jsou indikovány hodnotami výstupů PocetUdalosti Preplneni a Ztrata. Na závěr READC daný diskový soubor uzavře.

### Upozornění

Ve stávající implementaci funkce READC je možno jednorázově načíst z archivního souboru nejvýše 128 vzorků!

## 4.38 AKTC - aktualizace binárního archivního souboru

### Určení funkce a její charakteristika

Funkce AKTC zpracovává skupinu polí naplněných údaji data, času a příslušnými vzorky signálů typově odpovídajících archivovaným proměnným konkrétního volání funkce WRITEC se stejným archivním diskovým souborem, vyhledá podle časových údajů skupiny vzorků novějších než nejnovější záznam v souboru a ukládá do souboru nové záznamy až do vyčerpání rozsahu vstupních polí. Každý zapisovaný záznam přepisuje nejstarší záznam souboru (pokud již byl pořízen).

Funkce AKTC poskytuje - vedle funkce WRITEC - alternativní možnost zápisu do cyklického binárního souboru. AKTC doplňuje záznamy s časovým údajem tvořeným na základě vstupních parametrů, nikoli údajů systémového data a času jako WRITEC a může na rozdíl od WRITEC v jednom volání zapsat do souboru větší počet záznamů. Funkce AKTC najde uplatnění v situacích, kdy nadřazená stanice archivuje data získávaná ve víceméně nepravidelných intervalech komunikačními kanály z řídicích stanic, ve stanici Master lze daty přijímanými po různě velkých kvantech "krmit" pomocí AKTC archivní soubor. Podřízené stanice mohou provádět vlastní archivaci menšího rozsahu a

nadřazené stanici mohou podle jejích požadavků poskytovat různá "časová okna" vlastního archivu po částech určených velikostí vyrovnávacích pamětí (tvořených poli). AKTC umožňuje budovat distribuovaný systém archivace hodnot proměnných, který toleruje omezenou kapacitu přenosového kanálu, obtíže vyplývající z jeho druhu (veřejná telefonní síť) i různě dlouhé absence stanice Master na příjmu (užití téhož PC k různým účelům, např. pro probíhající vývojové práce, nebo dokonce jen občasné připojení PC pro postupné načtení archivovaných dat a jejich uložení do diskového souboru).

Funkce AKTC se nemůže v programu vyskytovat bez příslušného volání (byť formálního) funkce WRITEC se shodným jménem archivního souboru (podrobnosti viz popis funkce WRITEC).

### Vyvolání funkce AKTC

Funkce má proměnný počet N+2 vstupních parametrů a jediný výstupní parametr:

```
PoleDatum, PoleCas, Pole1, Pole2, . . . , PoleN > AKTC > "JmenoSouboru";
```

### Parametry funkce

PoleDatum - pole typu BYTE ARRAY[#M][#3] (rok-1900,měsíc,den)

PoleCas - pole typu BYTE ARRAY[#M][#3] (hodina,minuta,sekunda)

Pole1,Pole2,...,PoleN - jednorozměrná pole o M prvcích s typy shodnými s příslušnými parametry odpovídajícího volání funkce WRITEC.

"JmenoSouboru" - řetězcová konstanta určující jednak soubor, ze kterého má být čteno, jednak svazující volání AKTC s konkrétním voláním funkce WRITEC, v němž je určen rozsah souboru (max.počet záznamů),

N - počet vstupních polí, který musí souhlasit s počtem archivovaných signálů ve volání WRITEC, se nímž je dané volání AKTC sdruženo,

M - počet vstupujících "událostí", tj. skupin vzorků všech archivovaných signálů, z nichž každá je doprovázena časovým údajem v polích PoleDatum a PoleCas,

### Zpracování dat ve funkci AKTC

Funkce AKTC otevře existující soubor daného jména "JmenoSouboru", anebo soubor daného jména vytvoří. Každá trojice prvků polí PoleDatum a trojice prvků PoleCas je pro postupně rostoucí první indexy těchto polí převedena na jediný čtyřbytový údaj "počet sekund od začátku r.1993" a srovnána s časovým údajem posledního záznamu v archivním souboru. Počínaje prvním časovým údajem ležícím mimo časový interval dosud archivovaných dat, jsou vzorky ze vstupních polí Pole1 až PoleN zapisovány jako nové záznamy do souboru (zapisovaný záznam přepíše vždy nejstarší záznam v souboru) a zároveň je vždy zapsán do oblasti časových značek daného souboru časový údaj získaný převodem skupiny hodnot z polí PoleDatum a PoleCas. Takto probíhá aktualizace archivního souboru až do vyčerpání rozsahu vstupních polí popř. až do nalezení vstupní události s časem původu starším než nejnovější archivovaná událost. Na závěr AKTC daný diskový soubor uzavře.

## **4.39 LASTTC - načtení data a času posledního záznamu archiv.soub.**

### Určení funkce a její charakteristika

Funkce LASTTC slouží ke zjištění časového údaje posledního záznamu uloženého v archivním souboru.

### Vyvolání funkce LASTTC

Formát vyvolání funkce LASTTC je následující

```
"JmenoSouboru" > LASTTC > DatumDo, CasDo;
```

### Parametry funkce

"JmenoSouboru" - řetězcová konstanta určující soubor, ze kterého má být čten časový údaj posledního záznamu

DatumDo - pole typu BYTE ARRAY[#3] (rok-1900,měsíc,den)

CasDo - pole typu BYTE ARRAY[#3] (hodina,minuta,sekunda)

### Zpracování dat ve funkci AKTC

Funkce LASTTC naplní výstupní pole dvěma trojicemi údajů ve formátu \$DATE a \$TIME.

## 4.40 PUTARR, GETARR

### Určení funkce a její charakteristika

Funkce PUTARR a GETARR jsou použitelné u řídicích stanic řady TRONIC pro řízení výměny dat mezi programem a programovými driverly HW prostředků. Vzhledem k tomu, že většina driverů, vyvolávaných funkcemi PUTARR a GETARR závisí na hardware stanice, je nezbytné informovat se v technických příručkách jednotlivých řídicích stanic, zda je příslušný driver podporován firmwarem stanice.

#### Funkce PUTARR

slouží především k přenosu dat z uživatelského programu do daného zařízení. Formát volání funkce PUTARR je následující

```
VýstupníPole, TypZpracování > PUTARR > Výsledek;
```

kde

VýstupníPole je bytové pole, jehož délka závisí na konkrétním typu zpracování dat ve funkci

TypZpracování je hexadecimální číslo v rozsahu 0 až FF<sub>H</sub>

Výsledek je proměnná typu BOOL, její význam závisí na konkrétním typu zpracování

#### Funkce GETARR

slouží především k přenosu dat z daného zařízení do uživatelského programu. Formát volání funkce GETARR je následující

```
TypZpracování > GETARR > VstupníPole, Výsledek;
```

kde

TypZpracování je hexadecimální číslo v rozsahu 0 až FF<sub>H</sub>

VstupníPole je bytové pole, jehož délka závisí na konkrétním typu zpracování dat ve funkci

Výsledek je proměnná typu BOOL, její význam závisí na konkrétním typu zpracování

GETARR		PUTARR	
01 <sub>H</sub> ÷ EF <sub>H</sub>	příjem zpráv ze sběrnice AMICAN	01 <sub>H</sub> ÷ EF <sub>H</sub>	vysílání zpráv na sběrnici AMICAN
F0 <sub>H</sub>	čtení z archivu dat	F0 <sub>H</sub>	zápis do archivu dat
F1 <sub>H</sub>	univerzální zadávací místo displeje	F1 <sub>H</sub>	nastavení systémového času z uživ. progr.
F2 <sub>H</sub>		F2 <sub>H</sub>	uložení konstant do paměti FLASH
F3 <sub>H</sub>		F3 <sub>H</sub>	
F4 <sub>H</sub>		F4 <sub>H</sub>	výstup na tiskárnu (T2008S)
F5 <sub>H</sub>		F5 <sub>H</sub>	
F6 <sub>H</sub>		F6 <sub>H</sub>	
F7 <sub>H</sub>	čtení obsahu bufferu do pole	F7 <sub>H</sub>	uložení obsahu pole do bufferu
F8 <sub>H</sub>		F8 <sub>H</sub>	vyslání zprávy I na sběrnici MODBUS
F9 <sub>H</sub>	příjem zprávy ze sběrnice MODBUS	F9 <sub>H</sub>	vyslání zprávy II na sběrnici MODBUS
FA <sub>H</sub>		FA <sub>H</sub>	zrušení spojení přes modem
FB <sub>H</sub>	příjem zprávy přes modem	FB <sub>H</sub>	vyslání zprávy přes modem
FC <sub>H</sub>		FC <sub>H</sub>	navázání spojení přes modem

V následujících kapitolách jsou popsány jednotlivé procedury, prováděné funkcemi GETARR a PUTARR.

#### 4.40.1 Přenos dat po sběrnici AMICAN

Sběrnici CAN s protokolem AMICAN je možno komunikovat se všemi dodávanými stanicemi TRONIC. Potřebné hardwarové vybavení stanic a inicializace komunikačních driverů je uvedena příslušných technických příručkách.

Protokol AMICAN umožňuje vzájemné spojení procesních stanic TRONIC s dalšími stanicemi nebo PC. Všechny stanice jsou v přístupu ke sběrnici rovnocenné. Komunikace probíhá po blocích délky maximálně 250 bytů.

Odeslání bloku dat je provedeno zavoláním funkce PUTARR.

```
BlokDat, "XX" > PUTARR > Výsledek;
```

BlokDat je bytové pole následujícího formátu

délka (1 až 250)	blok vysílaných dat
------------------	---------------------

Sama délka se do počtu vysílaných bytů nezapočítává.

XX reprezentuje adresu přijímací stanice v rozsahu 1 až 240

Výstupní bitová proměnná Výsledek indikuje, zda se podařilo zprávu umístit do vysílací fronty. Pokud je nulová, výstupní fronta je plná a proto byla zpráva odmítnuta

Z vysílací fronty jsou zprávy postupně odesílány jednotlivým příjemcům. Přijímací stanice uloží přijatou zprávu do bufferu, jehož číslo je stejné, jako adresa vysílací stanice. Přijímací stanice se dotazuje na stav jednotlivých bufferů voláním funkce GETARR

```
"XX" > GETARR > BlokDat, Výsledek;
```

XX je číslo přijímacího bufferu

Výsledek udává, zda je v bufferu uložena přijatá zpráva, pokud ano (vrácená hodnota 1), je tato zpráva přenesena do pole BlokDat. V poli (stejně jako při vysílání funkcí PUTARR) udává první byte přijatou délkou, za ním následuje přijatá zpráva. Pokud je velikost pole menší, než délka přijaté zprávy, je tato zpráva zkrácena.

#### Priorita a pořadí zpráv, přenášených po sběrnici s protokolem AMICAN

Vzhledem k tomu, že sběrnice s protokolem AMICAN je typu PEER-TO-PEER, neexistuje ani MASTER, přiděluje sběrnici jednotlivým stanicím. Přidělování sběrnice je kolizní s řešením kolizí na principu priority zpráv. V praxi to

znamená, že zpráva s vyšší prioritou je po sběrnici přenesena dříve, než zpráva v nižší prioritou. Priorita zpráv závisí jednak na typu zprávy a u zpráv se stejným typem na adrese stanice, pro kterou je zpráva určena. Z hlediska uživatele jsou všechny zprávy stejného typu (typ DATA, viz. popis komunikačního protokolu AMICAN) a proto je pořadí zpráv určeno adresou přijímací stanice tak, že zpráva pro stanici s nižší adresou má vyšší prioritu na sběrnici.

### Zpracování dat v PUTARR, GETARR a navazující operace komunikačního systému

Aplikační program musí před vyvoláním funkce PUTARR nebo GETARR obsahovat deklaraci polí typu BYTE tvořících pomocné paměťové oblasti pro vysílané resp. přijaté zprávy (v předchozích příkladech BlokDat), a to pro všechny stanice projektu, se kterými je potřebné pomocí PUTARR a GETARR komunikovat.

Strukturu vysílaných a přijímaných zpráv určí deklarace dalších proměnných ležících ve stejném paměťovém prostoru - provádí se pomocí deklarace s EQU. Možné, a obvykle výhodnější, je celý soubor deklarací uspořádat tak, že nejprve deklarujeme pole typu BOOL (pokud požadujeme přenosy BOOL proměnných), přes něj (EQU) pole BlokDat a dále deklarujeme v prostoru tohoto pole všechny proměnné, jejichž hodnoty zamýšlíme poskytnout jiné stanici projektu. Jestliže je určitou skupinu proměnných potřebné sdílet v několika stanicích, deklarujeme je bezprostředně za sebou v oblasti zasažené požadovaným počtem polí pro zprávy. Blok deklarací s překrýváním pomocí EQU zpravidla využijeme jako základ pro deklarace pole BlokDat v programu stanice (stanie), pro kterou je zpráva určena.

Pokud nehrozí překročení maximální povolené délky zprávy, je pro jednoduchost výhodné unifikovat co největší část zpráv pro různé stanice, tedy vysílat některým stanicím i veličiny, které nebudou využity. Část prostoru polí zpráv může být překryta i vícenásobně pomocnými veličinami (obecně různých typů), které bude nutné před předáním zprávy komunikačnímu systému (voláním PUTARR) naplnit konkrétními daty exekučně (přířazovacími příkazy). V případě, že je potřebné vysílat větší počet BOOL proměnných, které z nějakého důvodu nemohou ležet pohromadě (např. jsou rozptýleny ve vstupním a výstupním poli) a je zároveň potřebné šetřit délkou zprávy, může být výhodné "koncentrovat" více bitů do proměnné BYTE či FIXP pomocí funkce CPS.

Aplikační program dále musí na vysílací straně zajistit synchronizaci s činností komunikačního systému. Vysílací driver obsahuje frontu pro čtyři vysílané zprávy. Vyvoláním PUTARR dojde k přenesení obsahu pole BlokDat do výstupní fronty driveru. Pokud je již vysílací fronta plná a zprávu se tedy do ní nepodařilo umístit, vrací funkce PUTARR proměnnou Vysledek = 0. Aplikační program musí potom opakovat pokus o odeslání zprávy v příštím cyklu. V praxi k tomu může dojít u zpráv s nižší prioritou přenosu po sběrnici

Stanice přijímající ukládá korektně přijaté zprávy do vyrovnávacích pamětí přiřazených jednotlivým stanicím projektu. Vyvolání funkce GETARR lze chápat jako výzvu k převzetí korektně přijaté zprávy - je-li přijatá zpráva k dispozici, je přenesena do pole BlokDat a je nastaven příslušný signál Výsledek; v opačném případě je pouze vynulován Výsledek a hodnoty všech proměnných alokovaných v prostoru pole BlokDat zůstávají zachovány. Pokud se v deklaracích nepodařilo "umístit" všechny požadované proměnné do prostoru pole BlokDat, je potřebné za vyvolání GETARR zařadit signálem Výsledek podmíněné zpracování zbývajících proměnných (přenesení přijatých hodnot ze zprávy do pracovních buněk - pro cílové BOOL proměnné se zde může uplatnit funkce DCPS). Pokud nejsou přijímaná data aktualizována dostatečně často, je asi vhodné vyhodnotit pravděpodobný výpadek vysílající stanice nebo poruchu sběrnice a učinit přiměřená opatření v části vlastního řídicího algoritmu (např.: !Vysledek, TLimit > DSR > SET > VýpadekVysílače;).

Souhrnně lze o používání funkcí PUTARR a GETARR pro výměnu datových polí mezi účastníky na sběrnici AMICAN konstatovat následující:

PUTARR voláme podmíněně při potřebě aktualizace hodnot v cizí stanici. Naproti tomu GETARR můžeme vyvolávat trvale periodicky, abychom v co nejkratším čase převzali přijatou zprávu. Proměnná Výsledek podmiňuje následné zpracování přijaté zprávy (přínejmenším rozeslání hodnot jednotlivých proměnných) a má také diagnostický význam.

#### 4.40.2 Paměť se sekvenčním přístupem pro záznamy dat

Ve stanicích TRONIC je k dispozici 44 až 64Kbyte paměti pro záznam dat. Kapacita paměti závisí na typu stanice, hodnoty kapacity paměti jsou uvedeny v technických příručkách jednotlivých stanic. U všech stanic je paměť zálohována baterií při výpadku napájení.

Do paměti se přenáší datový blok, který je uložen v jednorozměrném poli typu FIXP, které má následující formát

adresa v paměti	délka bloku	datový blok
-----------------	-------------	-------------

kde:

adresa v paměti je adresa FIXP (rozsah 0 až 32K-1)

délka bloku je čistá délka FIXP dat (rozsah 1 až 32K)

POZOR - oba údaje se vztahují k FIXP proměnným, paměť se tedy chová jako šestnáctibitová.

Po naplnění výstupního pole FIXP se volá funkce PUTARR s typem F<sub>0H</sub>, která přeneše data z výstupního pole do paměti stanice.

Pro čtení dat z paměti se obdobným způsobem používá funkce GETARR. Vstupní FIXP pole obsahuje stejně jako výstupní na prvních dvou prvcích adresu a délku. Na další adresy uloží funkce GETARR data z paměti.

### Vyvolání funkcí PUTARR a GETARR

Obě funkce mají pevný počet parametrů:

Vystup, "F0" > PUTARR > Vysledek

"F0" > GETARR > Vstup,Vysledek

BOOL proměnné Vysledek nemají v tomto případě význam a jejich hodnota se voláním funkce nezmění.

## 4.40.2 Univerzální zobrazovací a zadávací bod displeje

Tato funkce umožňuje zobrazit na displeji stanice hodnotu proměnné a nastavit její hodnotu s velmi malými nároky na využitý prostor uživatelského programu. Zobrazovaná a nastavovaná je hodnota proměnné FIXP. Zobrazení je v pevné řádové čárce, což znamená, že funkce umožňuje zobrazit desetinnou tečku na libovolné pozici zobrazení, ale hodnota je vždy celočíselná. Např. zvolíme-li formát 3 znaky před tečkou a 2 za tečkou (32) a vstupní proměnná bude mít hodnotu 12345, bude se na displeji zobrazovat údaj 123,45. Stejně pravidlo platí i pro zadávání. Např. zadané vstupní formáty 23 i 32 budou produkovat stejnou FIXP hodnotu při zadání 12,345 (formát 23) nebo 123,45 (formát 32).

Tato funkce v podstatě supluje standardní funkce OUTS a ATN, ale s podstatně menšími nároky na velikost uživatelského programu obzvláště v případě stanic bez číselných kláves, kde nastavování probíhá pomocí šipek po jednotlivých dekádách.

Zadávací bod displeje pracuje ve dvou režimech

**Typ zobrazování** – v tomto režimu funkce pouze vypisuje na displej číslo v zadaném formátu spolu s příslušnou fyzikální jednotkou.

**Typ zadávání** - v tomto režimu uživatel může pomocí číselných kláves nebo šipek nastavit požadovanou hodnotu proměnné.

Inicializace funkce zadávacího bodu :

"F1" > GETARR > Vstup,Vysledek

Bytové pole Vstup má následující formát

typ činnosti	formát	hodnota	pozice na displeji	fyz. jednotka
--------------	--------	---------	--------------------	---------------

Typ činnosti - BYTE, má hodnotu 1 pro zobrazování a 2 pro zadávání

Formát - BYTE, dekadické číslo xyz, kde hodnota stovek zadává, je-li zobrazení se znaménkem, hodnota desítek počet cifer před tečkou a hodnota jednotek počet cifer za tečkou. Např. hodnota 113 udává zobrazení ve formátu ±N,NNN. Údaj může mít nejvýše pět platných cifer, desetinnou tečku a znaménko. Pokud je hodnota čísla formátu 200, jedná se o formát hodin ve tvaru HH:MM

Hodnota - FIXP, hodnota pro zobrazení a prostor pro zadanou hodnotu. Je-li zvolen formát nastavení hodin, udává hodnota číslo minuty ve dni (DMIN). Maximální zobrazitelné a zadatelné číslo je 99hodin, 59minut.

Pozice - BYTE udávající znakovou pozici počátku zobrazovacího bodu (0 až 39 pro první řádek a 40 až 79 pro druhý řádek)

Fyz. jednotka - BYTE udávající typ fyzikální jednotky podle následující tabulky

číslo	jednotka	číslo	jednotka	číslo	jednotka	číslo	jednotka
0	není	8	m	16	kg/h	24	MW
1	°C	9	l	17	l/h	25	rezerva
2	%	10	hl	18	l/m	26	rezerva
3	Pa	11	m <sup>3</sup>	19	l/s	27	rezerva
4	kPa	12	kg	20	sec	28	rezerva
5	MPa	13	t	21	min	29	rezerva
6	mm	14	m <sup>3</sup> /h	22	hod	30	rezerva
7	cm	15	t/h	23	GJ	31	rezerva

BOOL proměnná Vysledek má význam pouze v režimu zadávání, kde indikuje ukončení zadávání.

V režimu zadávání se na displeji objeví číselný údaj podle zadaného formátu s hodnotou, odpovídající požadované hodnotě v poli Vstup. Současně se na prvním zobrazovaném znaku objeví kurzor.

Pokud má klávesnice stanice číselné klávesy, probíhá nastavování pomocí těchto kláves Šípkami doprava a doleva je možné se přesunovat mezi jednotlivými dekádami údaje. Pokud kurzor pomocí šipek opustí zadávací pole, je zadávání zrušeno beze změny hodnoty v poli Vstup. Stiskem klávesy ACK se zadaná hodnota přepíše do pole vstup. V obou případech dojde k nastavení bitu Vysledek.

Pokud stanice číselné klávesy nemá, probíhá nastavování pomocí šipek. šípkami doleva a doprava se přechází kurzorem mezi jednotlivými dekádami, hodnota příslušné dekády se nastavuje šípkami nahoru a dolů. Pokud kurzor pomocí šipek opustí zadávací pole, je zadávání zrušeno beze změny hodnoty v poli Vstup. Stiskem klávesy ENTER se zadaná hodnota přepíše do pole vstup. V obou případech dojde k nastavení bitu Vysledek.

Po dobu trvání režimu zadávání nemá uživatelský program přístup k žádné klávese klávesnice stanice. Z uživatelského programu lze ukončit zadávání pouze přechodem do režimu zobrazování (voláním funkce GETARR s hodnotou 1 v prvním bytu pole Vstup. Znovunastartování režimu zadávání se neprovede, volání funkce GETARR bude v tomto případě ignorováno. To umožňuje cyklické volání funkce pro zobrazování i nastavení, což zjednodušuje ovládací uživatelský program.

### 4.40.3 Nastavení systémového času z uživatelského programu

Pro nastavení systémového času se užívá funkce PUTARR s adresou F1<sub>H</sub>. Tato funkce přenesení data z bytového pole do obvodu RTC.

#### Vyvolání funkce PUTARR

CasovePole, "F1" > PUTARR > Vysledek

#### Parametry funkce

CasovePole - bytové pole, které se bude vysílat

Vysledek - bitová proměnná, která je deklarována a použita pouze pro kompatibilitu funkce PUTARR, jeho hodnota se funkcí nemění.

## Zpracování dat v PUTARR

Nejprve se do bytového pole uloží časový údaj a dále se volá funkce PUTARR s adresou F1.

```
STAT BYTE CasovePole[0..6];
/*vteřiny v rozsahu 0 až 59*/      36 > CasovePole[0] ;
/*minuty v rozsahu 0 až 59*/      29 > CasovePole[1] ;
/*hodiny v rozsahu 0 až 23*/      15 > CasovePole[2] ;
/*den v týdnu v rozsahu 0 až 6*/   03 > CasovePole[3] ;
/*den v rozsahu 1 až 31*/         14 > CasovePole[4] ;
/*měsíc v rozsahu 1 až 12*/       10 > CasovePole[5] ;
/*rok v rozsahu 0 až 99 */        99 > CasovePole[6] ;
CasovePole, "F1" > PUTARR > Uspech;
```

### 4.40.4 Uložení konstant do paměti FLASH ROM

Konstanty se do paměti FLASH ROM ukládají funkcí PUTARR s adresou F2<sub>H</sub>

#### Vyvolání funkce PUTARR

```
Pole, "F2" > PUTARR > Uspech;
```

#### Parametry funkce

Pole - libovolné pole s libovolnou délkou, jeho obsah nemá pro toto volání funkce PUTARR význam a nemění se.

Vysledek - proměnná, která je nastavena, pokud ukládání proběhlo úspěšně. Vzhledem k tomu, že po dobu ukládání dat je pozastaveno vykonávání programu, je možno testovat tento bit bezprostředně po volání funkce PUTARR.

### 4.40.5 Ovládání paralelní tiskárny (pouze T2008S)

Paralelní tiskárna se připojuje ke stanici T2008S pomocí expanzního modulu ELPT, který se zasunuje do konektoru P2 na základní desce nebo posledním expanzním modulu. Tiskárna se připojuje pomocí standardního kabelu pro spojení tiskárny a PC. **Při připojování musí být jak stanice tak i tiskárna vypnuty.**

#### Vyvolání funkce PUTARR

```
Tisk, "F4" > PUTARR > UspechTisk
```

#### Parametry funkce

Tisk - pole dat, které se vysílá do tiskárny, první byte pole udává počet vysílaných bytů

UspechTisk - bitová proměnná, která se funkcí PUTARR nuluje a nastavuje se asynchronně po úspěšném vyslání dat do tiskárny.

### 4.40.6 Uložení a vytažení dat z bufferu paměti

Tyto funkce jsou určeny pro rychlé přenosy dat mezi dvěma poli a nahrazují přenos pomocí FOR cyklu, který je relativně pomalý. Funkce jsou jistou obdobou nakopírování řetězce do clipboardu a zpětného uložení obsahu clipboardu do proměnné. Formát volání funkcí je následující:

```
Pole1, "F7" > PUTARR > Vysledek;
"F7" > GETARR > Pole2, Vysledek;
```

Pomocí funkce PUTARR se do bufferu paměti uloží data z bytového pole Pole1. Maximální délka záznamu je 256 bytů. Počet uložených bytů je určen deklarovanou velikostí Pole1. Proměnná Vysledek nemá v tomto případě význam a její hodnota se nemění.

Pomocí funkce GETARR se vyčtou data z bufferu a uloží se do pole Pole2. Počet vyčtených dat je určen zaznamenanou délkou při předchozím uložení. Pokud je velikost pole Pole2 větší nebo rovna délce záznamu, je uloženo tolik dat, kolik je zaznamenaná délka a proměnná Vysledek je nastavena na hodnotu 1. Pokud je velikost Pole2 menší, než délka záznamu, uloží se pouze tolik bytů, kolik je velikost pole, v tomto případě je hodnota proměnné Vysledek nastavena na 0.

#### 4.40.6 Komunikace po telefonních linkách pomocí modemů

Pomocí modemu lze komunikovat jak mezi stanicemi navzájem, tak i mezi stanicí a PC.

Komunikace s modemem probíhá pomocí standardních prostředků jazyka LEDA, funkcemi PUTARR a GETARR. Ukončení každé operace je indikováno nastavením bitu Uspech, adresovaným funkcí, její výsledek je označen obsahem bytu vysledek, který je uložen v poli IL na adrese FFH. Pokud dojde k navázání spojení, je identifikátor stanice, s níž bylo spojení navázáno v poli IL na adrese FEH. Pokud není navázáno spojení s žádnou stanicí, je v tomto bytu hodnota 0.

Ovladač modemu se může nacházet v jednom z pěti stavů :

- NO MODEM** - modem nebyl inicializován
- STANDBY** - modem prošel úspěšně inicializací
- CONNECT** - bylo navázáno spojení mezi modemy
- CONFIRM** - potvrzeno ID volané stanice
- SHUTDOWN** - zrušení spojení

V následující tabulce jsou významy jednotlivých hodnot bytu výsledek.

Hodnota	význam bytu FF <sub>H</sub> – výsledek předchozí operace
0	funkce proběhla úspěšně
1	chyba při inicializaci
2	chyba při navazování spojení - modemy se nespojily
3	chyba při navazování komunikace – nesouhlasí ID
4	délka přijaté zprávy byla delší než délka pole funkce GETARR
5	délka vysílané zprávy je větší než délka komunikačního bufferu
6	nepodařilo se přenést zprávu, spojení bylo zrušeno
7	ve stavu CONNECT přišel znak result codu modemu

#### Inicializace driveru

Ovladač modemu se inicializuje řetězcem v úvodní hlavičce programu v jazyce LEDA. Řetězcem se vybírá příslušný driver modemu, určuje komunikační rychlost mezi stanicí a modemem a určuje se číslo stanice pro případnou identifikaci. CFG řetězec má následující formát:

```
CFG "MOD:ID";
```

kde ID je identifikační číslo stanice v hexadecimálním formátu, nesmí být =0.

Identifikační číslo stanice slouží k verifikaci stanice po navázání spojení. Po inicializaci ovladače dochází k inicializaci modemu, pokud je inicializace úspěšná, nastaví výsledek do nuly. Pokud není úspěšná, ovladač nastaví výsledek roven 1 a pokus o inicializaci se opakuje.

## Navázání spojení

Spojení lze navazovat pouze pokud je ovladač ve stavu STANDBY. V jakémkoli jiném stavu je tato funkce ignorována. Se vzdálenou stanicí se navazuje spojení funkcí PUTARR(00H) podle následujícího vzoru:

```
Pole, "F8" > PUTARR > Uspech;
```

kde:

Pole - bytové pole s následujícím formátem {ID,TELNUM},

ID - identifikátor stanice, s nímž se navazuje spojení,

TELNUM - telefonní číslo stanice, s nímž navazujeme spojení, toto číslo je ukončeno znakem CR. Uvnitř čísla mohou být všechny znaky povolené protokolem modemu, celé číslo se do modemu přenáší v nezměněné podobě.

Po navázání spojení stanice nejprve verifikuje číslo protější stanice s požadovaným ID. Po ukončení operace je nastaven bit Uspech a aktualizuje se hodnota výsledku v poli IL podle tabulky výsledků.

## Ukončení spojení/reinicializace modemu

Pokud se ovladač nachází ve stavu CONNECT, dojde k přerušení spojení, pokud je v jakémkoli jiném stavu, dojde k reinicializaci modemu. Spojení se ukončuje spojení funkcí PUTARR(02H) podle následujícího vzoru

```
Pole, "FA" > PUTARR > Uspech;
```

Pole v tomto případě je pouze formální, nenese žádnou informaci. Zavoláním této funkce dojde k zrušení spojení, rozpojení modemů a uvedení ovladačů obou stanic do základního stavu.

## Vyslání pole do modemu

Pole je možno přenášet pouze ve stavu CONFIRM, v ostatních stavech je funkce ignorována. Pole se do modemu vysílá funkcí PUTARR(01H) podle následujícího vzoru

```
Pole, "F9" > PUTARR > Uspech;
```

kde:

Pole - bytové pole s následujícím formátem {NUM,DATA},

NUM - délka vysílané zprávy (slovo),

DATA - vysílaná zpráva s maximální délkou 1024 bytů.

Po ukončení vysílání je nastaven bit Uspech a aktualizována hodnota výsledku.

## Příjem zprávy z modemu

Zpráva z modemu se přijímá automaticky, pokud se ovladač nachází ve stavu CONNECT. V tomto stavu se přijatá zpráva ukládá do bufferu odkud je vyzvednuta funkcí GETARR(01H)

```
"F9" > GETARR > Pole, Uspech;
```

kde:

Pole - je bytové pole, do kterého je přenesena nevyzvednutá zpráva z komunikačního bufferu. První dva byty obsahují délku zprávy a dále následují přijatá data. Pokud je délka přijaté zprávy větší než délka pole, je zpráva příslušně zkrácena.

Uspech - BOOL proměnná.

## Omezení přenosu

Pokud není ve stavu CONNECT přenesena žádná zpráva po dobu 5 minut, je spojení automaticky ukončeno a ovladač přechází do stavu STANDBY. Maximální délka setrvání ovladače ve stavu CONNECT je 1 hodina, po této době je spojení ukončeno za všech okolností. Ve stavu STANDBY je modem reinicializován každých 5 minut.

#### 4.40.7 Komunikace po sběrnici MODBUS RTU

Sběrnice MODBUS umožňuje připojení jedné nebo několika stanic k PC. Přidělování sběrnice je typu MASTER – SLAVE, kde masterem je stanice PC a všechny stanice TRONIC jsou typu slave. Komunikace probíhá po sběrnici RS485.

##### Inicializace:

Komunikační driver stanice se inicializuje buďto pomocí CFG řetězce nebo ze systémového menu stanice. Způsob inicializace konkrétního typu stanice je uveden v technické příručce příslušné stanice.

Inicializace pomocí CFG řetězce:

V hlavičce programu je uvedeno klíčové slovo CFG a za ním příslušné inicializační parametry

```
CFG "MDB:BP, ID";
```

kde:

MDB je klíčové slovo pro typ komunikace po sběrnici MODBUS

B je kód přenosové rychlosti(0 – 2,4kBd, 1 – 4,8kBd, 2 – 9,6kBd,3 – 19,2kBd)

P - kód parity (0 – bez parity, 1 – sudá parita, 3 – lichá parita)

ID - identifikační číslo stanice v hexadecimálním formátu v rozsahu <1,FF>

Stanice komunikuje s masterem pomocí dvou typů zpráv - Data transfer a Auto response.

##### Zpráva typu Data Transfer

Přenos dat začíná stanice MASTER vysláním zprávy typu DATA. Stanice cyklicky testuje příchod této zprávy pomocí funkce

```
"F9" > GETARR > Pole, Uspech;
```

kde:

Pole - přijímací BYTE pole příslušné délky. Buňka Pole[0] udává počet přijatých bytů.

Uspech - BOOL proměnná, která indikuje přijatou zprávu, která je umístěna do Pole.

Uživatelský program vyhodnotí význam přijaté zprávy (závisí na uživateli) a může odeslat odpověď do stanice MASTER pomocí funkce PUTARR

```
Pole, "F9" > PUTARR > Uspech;
```

kde:

Pole - BYTE pole pro vysílaná data, v prvním bytu Pole[0] je uložena délka vysílaných dat. Maximální délka vysílaných dat je 250 bytů.

Uspech - BOOL proměnná, která indikuje úspěšné vyslání datového pole, nastavuje se po jeho odeslání.

##### Zpráva typu Auto Response

Vzhledem k tomu, že komunikace zprávami typu DATA TRANSFER vyžaduje pro přenos ve směru SLAVE -> MASTER reakci uživatelského programu která závisí na periodě zpracování obslužné sítě komunikace, dochází při přenosech dotaz – odpověď k značným prodlevám na sběrnici. To by mohlo v rozsáhlých systémech způsobovat značné zpomalování komunikace. Tato zpoždění mohou být eliminována použitím zpráv typu AUTO RESPONSE. Tyto zprávy slouží k přenosu bloku dat ze stanice SLAVE do MASTER. Uživatelský program předem připraví vysílanou zprávu do bufferu pro Auto Response. Obsah tohoto bufferu je potom asynchronně přenesen do stanice MASTER jako odpověď na dotazovou zprávu REQUEST. Do vysílacího bufferu se zpráva přenáší funkcí

```
Pole, "F8" > PUTARR > Uspech;
```

kde:

Pole - přijímací BYTE pole příslušné délky. Buňka Pole[0] udává počet přijatých bytů.

Uspech - BOOL proměnná, která indikuje přijatou zprávu, která je umístěna do Pole.

Vzhledem k tomu, že přenos zprávy typu AUTO RESPONSE je zcela asynchronní, je možno cyklicky vysílat pouze jednu zprávu (z hlediska struktury zprávy). Proto je při programování komunikace vhodné umístit do této zprávy ty proměnné, které je zapotřebí cyklicky zpracovávat v nadřazené stanici (např. stavy vstupů, rychle se měnících vnitřních proměnných atd.). Ostatní proměnné které stanice MASTER nepotřebuje číst tak často, je možné přenášet pomocí zpráv DATA TRANSFER.

## 5. Popis standardních reprezentací

Standardní reprezentace jazyka LEDA jsou vlastně - jak již bylo uvedeno výše - standardními funkcemi, které bez ohledu na umístění v programu (v síti interpretované v procesní stanici nebo v MASTER síti interpretované v operátorské stanici) zpracovává program TOLEDA běžící v PC. Reprezentace zajišťují prezentaci hodnot libovolných proměnných programu na obrazovce PC v textové nebo grafické formě a zpracování textů zadávaných za běhu programu operátorem.

### 5.1 Standardní reprezentace bezrozměrných veličin

Pro zobrazování a zadávání hodnot proměnných typu BYTE resp. FIXP v procentech "strojové jednotky" (250 resp. 8000) slouží standardní reprezentace s identifikátorem "BP", resp. "FP". Tyto dvouznakové identifikátory se prakticky používají pouze v situaci, kdy má být identifikátor reprezentace parametrem standardní funkce - např. u funkcí ATN, OUTS a OUTF. Ve skalárních parametrech je povoleno psát např. místo dvojice znaků \$FP popř. FP\$ pouze samotný znak \$, např.:

```
VHV1 > $ _____;
```

Reprezentace \$FP zobrazuje FIXP čísla ve tvaru "SNNN.N%"; reprezentace \$BP zobrazuje hodnoty BYTE proměnných ve formátu "NNN.N%" (S je mezera nebo znaménko "-"), avšak při generaci znakových polí pomocí OUTS a OUTF se čísla zkracují nahrazením případné úvodní nuly mezerou a vypuštěním desetinné tečky, následující číslice a symbolu %. Všechny standardní reprezentace uvedené dále v následujícím odstavci se při použití v rámci generace výstupních řetězců určených k tisku, zápisu do diskových souborů nebo k vyslání prostřednictvím sériového portu chovají podobně jako reprezentace \$FP, resp.\$BP, tzn., že nahrazují úvodní nuly mezerami a vypouštějí symboly fyzikálních jednotek.

Pokud není zadán identifikátor reprezentace a ve skalárních parametrech není uveden ani symbol "\$", zobrazují se a zadávají se číselné údaje bez jakýchkoli přídatných znaků a ve standardních formátech čísel typu BYTE, FIXP a FLTP, tj. čísla:

BYTE ve formátu "NNN" v rozsahu 0 až 255 (bez znaménka),

FIXP ve formátu "SNNNNN" v rozsahu -32768 až 32767,

FLTP ve formátu "SNNNNN.NNN " (poslední znak je mezera) s pohyblivou desetinnou tečkou pro čísla v rozsahu absolutních hodnot od 0.0001000 do 99999999. nebo v exponenciálním tvaru s fixovanou polohou desetinné tečky za první cifrou mantisy "SN.NNNNESNN" pro čísla v absolutní hodnotě menší nebo větší, avšak vždy v celkové délce jedenácti znaků.

Je-li nutné zadat použití těchto standardních reprezentací jako parametru ve volání standardních funkcí - např. ATN, OUTS a OUTF, je nutné explicitně zadat identifikátor reprezentace jako BY pro BYTE proměnné nebo FI pro FIXP proměnné nebo FL pro FLTP proměnné. Ve skalárních parametrech je použití identifikátorů reprezentací BY, FI a FL nepovinné.

Hodnoty proměnných typu BYTE nebo FIXP je možné zobrazovat a zadávat také jako hexadecimální nebo binární čísla.

Pro proměnné BYTE je určena reprezentace \$X zobrazující dvouciferná hexadecimální čísla a reprezentace \$B zobrazující řetězec osmi binárních číslic (bez mezer).

Pro proměnné FIXP je určena reprezentace \$XX zobrazující čtyřciferná hexadecimální čísla a reprezentace \$BB zobrazující řetězec šestnácti binárních číslic.

## 5.2 Standardní reprezentace fyzikálních veličin

Pro některé často používané typy čidel a převodníků jsou k dispozici standardní reprezentace převádějící hodnoty veličin typu FIXP na zobrazovaný číselný údaj doplněný řetězcem znaků příslušného fyzikálního symbolu a obráceně pro převod zadávaného řetězce na hodnotu proměnné typu FIXP.

Pro průtoky a tlaky se jedná o následující reprezentace s lineární charakteristikou (vstupem je zpravidla unifikovaný proudový signál):

\$Identifikátor reprezentace	základní rozsah	fyzikální veličina	základní rozsah zobraz.čísel a symbol fyzikální jednotky
\$F1K	0 až 8190	průtok	0 až 1000.0 m <sup>3</sup> /hod
\$FK4	0 až 8190	průtok	0 až 400.0 m <sup>3</sup> /hod
\$FK25	0 až 8190	průtok	0 až 250.0 m <sup>3</sup> /hod
\$FK16	0 až 8190	průtok	0 až 160.0 m <sup>3</sup> /hod
\$P4M	-8190 až 8190	tlak	-4.000 až 4.000 MPa
\$P1M6	-8190 až 8190	tlak	-1.600 až 1.600 MPa

Horní hranici základního rozsahu odpovídá maximum výstupu A/D převodníku, tj. proudový signál 20mA. V programu se však mohou vyskytovat hodnoty až 4krát větší a standardní reprezentace je také - s omezením daným počtem číslic - zobrazí. Naopak zadávání fyzikálních hodnot ležících mimo uvedený základní rozsah reprezentace se vyhodnocuje jako chyba.

Pro odporové teploměry Pt100 jsou k dispozici dvě standardní reprezentace zajišťující linearizaci:

\$Identifikátor reprezentace	základní rozsah	fyzikální veličina	základní rozsah zobraz.čísel a symbol fyzikální jednotky
\$T2	0 až 8190	teplota	-050.0 až 200.0°C
\$T5	0 až 8190	teplota	000.0 až 500.0°C

Linearizace přitom probíhá podle tabulky (s lineární interpolací):

t [°C]	R[Ohm]	bity \$T5	bity \$T2
-50	80.25	----	0000
0	100.0	0000	1653
50	119.40	0858	3274
100	138.50	1702	4875
150	157.32	2534	6450
200	175.84	3353	8000
250	194.08	4160	----
300	212.03	4954	----
350	229.69	5734	----
400	247.06	6502	----
450	264.14	7258	----
500	280.93	8000	----

V posledním sloupci "bity" jsou hodnoty pro jmenovitý rozsah A/D převodníku 0 až 8000 bitů, který lze považovat za doporučený pro řídicí stanice, na nichž má být provozován interpret jazyka LEDA. Reprezentace \$T2 a \$T5 nejsou mimo uvedený rozsah použitelné - na mezních teplotách dochází k saturaci. Pro případy nepokryté uvedenými standardními reprezentacemi a dále také v situaci, kdy je potřebné např. na teplotách provádět různé aritmetické operace, je vhodné zajistit v rámci vstupního zpracování analogových vstupů (bezprostředně za funkcí [IPR](#)) takový přepočít (obecně linearizační - např. s využitím [NLT](#)), který umožní použití některé z následujících standardních reprezentací pro FIXP proměnné:

\$Identifikátor reprezentace	základní rozsah	fyzikální veličina	základní rozsah zobraz. čísel a symbol fyzikální jednotky
\$TT	-9999 až 9999	teplota	-999.9 až 999.9 °C
\$PP	-9999 až 9999	tlak	-9.999 až 9.999 MPa
\$FF	-9999 až 9999	průtok obj.	-999.9 až 999.9 m <sup>3</sup> /h
\$MM	-9999 až 9999	průtok hm.	-999.9 až 999.9 t/h
\$EE	-9999 až 9999	energie	-999.9 až 999.9 GJ
\$WW	-9999 až 9999	výkon	-99.99 až 99.99 MW
\$LL	-9999 až 9999	hladina	-999.9 až 999.9 cm
\$QQ	-9999 až 9999	objem	-999.9 až 999.9 m <sup>3</sup>
\$GG	-9999 až 9999	hmotnost	-999.9 až 999.9 t

Při vybočení z uvedeného základního rozsahu je automaticky změněn formát zobrazení posunem desetinné tečky o jednu pozici doprava, to znamená, že je možné zobrazovat, popř. i zadávat, např. energie až do cca 3276GJ, objemy do 3276m<sup>3</sup> a hmotnosti do 3276t. Při požadovaném větším rozsahu a rozlišovací schopnosti lze doporučit - i za cenu zvýšení nároků na čas procesoru - přechod na používání veličin typu FLTP.

Ke každé z výše uvedených standardních reprezentací existuje reprezentace s ní sdružená, která zajišťuje pouze příslušný přepočít, ale nepřevádí výslednou hodnotu na zobrazovaný řetězec. Tato reprezentace tedy má vstupní i výstupní typ FIXP. Její identifikátor se používá jako skutečný parametr ve volání standardních reprezentací s grafickým výstupem (\$BAR, \$TIGR) a je odvozen od identifikátoru výchozí reprezentace doplněním o znak "\_" (podtržítka). Vzniknou tak tyto identifikátory:

P\_, T\_2, T\_5, F\_1K, F\_K16, F\_K4, F\_K25, P\_4M, P\_1M6,

TT\_, PP\_, FF\_, MM\_, EE\_, WW\_, LL\_, QQ\_, GG\_

Pro zpracování vstupů operátora (číselných konstant) zadávaných na místa určená markery lze používat uvedené reprezentace s tím, že formát vstupujícího čísla je obecně volným formátem čísel v pohyblivé řádové čárce. Sejmuté číslo je náležitě upraveno násobením příslušnou mocninou deseti (např. \*10 pro reprezentaci TT, \*100 pro reprezentaci WW, ..) a zanedbáním nadbytečných desetinných míst. Formáty vyvolání jsou uvedeny podrobně v kapitole 6.

### 5.3 Reprezentace pro formátování číselných údajů

Formáty vystupujících znakových řetězců číselných údajů - pro zobrazování, tisky a archivace v textových souborech - může uživatel určovat pomocí speciálních standardních reprezentací dvou skupin, s identifikátory:

FSMN

nebo

FUMN

přičemž:

S ... je znak uvozující reprezentaci se znakem znaménka

U ... je znak uvozující reprezentaci bez znaku znaménka

M ... je číslice udávající požadovaný počet číslic před des.tečkou

N ... je číslice udávající požadovaný počet číslic za des.tečkou

Pro FLTP proměnné je interpretace zadaného formátu přímočará, poznamenejme jen, že v situaci, kdy je vstupující číslo v absolutní hodnotě větší než maximálně zobrazitelné pro dané M, dojde k posunu desetinné tečky směrem doprava tak, že celkový počet cifer zůstává zachován na hodnotě M + N. Je-li naopak číslo menší, než odpovídá velikosti M, je výstupní řetězec zleva doplněn příslušným počtem nul, to znamená, že poloha znaménka pro reprezentace FSMN není závislá na velikosti čísla. Má tedy pro FLTP proměnné číslo M vlastně význam minimálního počtu číslic před desetinnou tečkou a N význam maximálního počtu číslic za tečkou.

Pro FIXP a BYTE proměnné mají číslice M a N poněkud jiný význam. Desetinná tečka je vlastně vsunuta mezi číslice výsledku (před N-tou zprava). Pro FIXP má smysl nejvýše  $M = 5$ ,  $N = 5$  a musí platit  $M + N \leq 5$ ; pro proměnné typu BYTE musí být  $M \leq 3$ ,  $N \leq 3$ ,  $M + N \leq 3$ .

## 5.4 \$D - zobrazení hodnot v digitální formě

### Určení reprezentace a její charakteristika

Standardní reprezentace \$D je určena k zobrazování číselných údajů ve formě znakových řetězců s eventuálním označením fyzikální jednotky, a to ve zvolené barvě na barevném pozadí a s určeným typem písma včetně jeho velikosti. Parametrem reprezentace \$D je také identifikátor reprezentace s výstupním typem CHAR - tato reprezentace může stanovit nelineární převod mezi vstupním signálem a hodnotou převáděnou na zobrazované číslo a dále formát tohoto čísla a řetězcovou konstantu symbolu fyzikální jednotky. Reprezentace \$D umožňuje také zobrazování data (rok-měsíc-den) resp. časového údaje (hodina: minuta:sekunda) - parametrem volání musí v tomto případě být reprezentace DATE resp. TIME.

### Formát volání \$D

Reprezentace \$D má tento pevný formát volání:

```
REP Vstup, Reprez., BarvaPísma, BarvaPozadí, Typ > $D > \MARK\;
```

### Parametry reprezentace

Vstup - proměnná typu BYTE | FIXP | FLTP

Reprezentace - identifikátor reprezentace se vstupním typem odpovídajícím výrazu Vstup a s výstupem typu CHAR popř. reprezentace DATE resp. TIME pro Vstup typu BYTE nebo BYTE ARRAY (trojice byte s významem rok, měsíc, den resp. hodina, minuta, sekunda)

BarvaPísma - konstanta typu BYTE určující barvu zobrazovaného řetězce

BarvaPozadí - konstanta typu BYTE určující barvu pozadí

Typ - konstanta typu BYTE určující formát zobrazení čísel

\MARK\ - odkaz na marker určující polohu prvního zobrazovaného znaku

### Formáty zobrazení čísel

Je-li zadána reprezentace T\_2 určená pro zobrazení teplot z odpor.teploměrů v rozsahu -50 až 200°C, potom se podle zadaného typu zobrazuje hodnota vstupní proměnné v jednom z následujících formátů:

typ:	formát zobrazení:
0	-999.9 °C
1	999.9 °C
2	-999 °C
3	999 °C
4	-99.9 °C
5	99.9 °C
6	-99 °C
7	99 °C

Není-li zadán identifikátor reprezentace T\_2, potom se podle zadaného typu zobrazuje takto:

typ:	formát zobrazení:	komentář
0	-32767	FIXP - pět cifer se znaménkem
1	-409.5%	FIXPPROC - FIXP v procentech, se znaménkem, 8000 odpovídá 100%
2	99%	FIXPPROC - dvě cifry bez znaménka
3	-99%	FIXPPROC - dvě cifry se znaménkem
4	-99.9%	FIXPPROC - tři cifry, jedno desetinné místo se znaménkem
5	9	FIXP - jedna cifra bez znaménka
6	99	FIXP - dvě cifry bez znaménka
7	999	FIXP - tři cifry bez znaménka

## 5.5 \$CD - zobrazení hodnot v digitální formě s řízenou barvou

### Určení reprezentace a její charakteristika

Standardní reprezentace \$CD je určena k zobrazování číselných údajů ve formě znakových řetězců s eventuálním označením fyzikální jednotky na barevném pozadí a v barvě určované momentální hodnotou vstupního parametru tvořící index do některé "palety barev" určené dalším parametrem. Parametrem reprezentace \$CD je také identifikátor reprezentace s výstupním typem CHAR - tato reprezentace může stanovit nelineární převod mezi vstupním signálem a hodnotou převáděnou na zobrazované číslo a dále formát tohoto čísla a řetězcovou konstantu symbolu fyzikální jednotky. Reprezentace \$CD je použitelná navíc také pro zobrazování údajů data resp. času pomocí reprezentací DATE resp. TIME (ve formátu rok-měsíc-den resp. hodina:minuta:sekunda).

### Formát volání \$CD

Reprezentace \$CD má tento pevný formát volání:

```
REP Vstup, Reprezentace, Barva, Paleta, Typ > $CD > \MARK\;
```

### Parametry reprezentace

Vstup - proměnná typu BYTE | FIXP | FLTP

Reprezentace - identifikátor reprezentace se vstupním typem odpovídajícím výrazu Vstup a s výstupem typu CHAR, popř. reprezentace DATE nebo TIME pro Vstup typu BYTE nebo BYTE ARRAY (jde o trojice byte s významem rok, měsíc, den resp. hodina, minuta, sekunda).

Barva - proměnná typu BYTE tvořící index do barevné palety

Paleta - identifikátor pole typu BYTE

Typ - konstanta typu BYTE určující formát výstupního řetězce

\MARK\ - odkaz na marker určující polohu prvního zobrazovaného znaku

Pole Paleta je potřebné naplnit čísly kódů barev, a to ve dvojicích (indexovaných výrazem Barva):

barva zobrazovaných znaků, barva pozadí

Barvy se zadávají ve formě čísel, která odpovídají kódování barev používanému fy BORLAND (pro grafické adaptéry EGA a VGA). Lze doporučit zavedení ekvivalentních řetězců, které usnadní zadávání barev ve všech reprezentacích resp. barevných paletách. Skupina ekvivalencí může vypadat např. takto:

"00" == "BLACK"	(černá)
"01" == "BLUE"	(modrá)
"02" == "GREEN"	(zelená)
"03" == "CYAN"	(modrozelená)
"04" == "RED"	(červená)
"05" == "MAGENTA"	(fialová)
"20" == "BROWN"	(hnědá)
"07" == "LIGHTGRAY"	(světle šedá)
"56" == "DARKGRAY"	(tmavě šedá)
"57" == "LIGHTBLUE"	(světle modrá)
"58" == "LIGHTGREEN"	(světle zelená)
"59" == "LIGHTCYAN"	(světle zelenomodrá)
"60" == "LIGHTRED"	(světle červená)
"61" == "LIGHTMAGENTA"	(světle fialová)
"60" == "YELLOW"	(žlutá)
"63" == "WHITE"	(bílá)

Pro parametr Typ platí totéž, jako pro reprezentaci \$D.

## 5.6 \$BAR - sloupcový graf

### Určení reprezentace a její charakteristika

Standardní reprezentace BAR umožňuje zobrazovat vertikální nebo horizontální sloupcové grafy. Sloupcový graf je charakterizován výškou, šířkou (v počtu pixelů), barvou pozadí základní obdélníkové plochy a dále barvou vlastního sloupce - obdélníka, jehož vertikální nebo horizontální rozměr je ovlivňován momentální hodnotou vstupní proměnné (obecně výrazu). Zcela obdobně jako u standardní reprezentace TIGR je možné před výpočtem velikosti sloupců předzpracovat hodnotu vstupní proměnné přes reprezentaci, jejíž identifikátor je jedním z parametrů BAR.

Reprezentace BAR umožňuje zpracovávat kromě vstupní proměnné obecně až čtyři další signály s významem dvojice varovných mezí a dvojice havarijních mezí. Při vybočení hodnoty vstupní proměnné z momentálního rozsahu varovných mezí dojde k přebarvení celého sloupce standardně na žlutou barvu, při vybočení této hodnoty z momentálního rozsahu havarijních mezí dojde k přebarvení sloupce na barvu červenou. Momentální hodnoty všech zadaných mezí jsou na příslušném místě základní obdélníkové plochy trvale zobrazovány barevnými úsečkami.

### Formát volání \$BAR

Volání reprezentace BAR má pevný formát:

```
REP Vstup, Reprezentace, Barva, "V"|"H", Výška, Šířka, Pozadí,
Min_Val, Max_Val, LoWar, HiWar, LoAlm, HiAlm > $BAR > \MARK\;
```

### Parametry reprezentace

Vstup - proměnná typu BYTE | FIXP | FLTP

Reprezentace - identifikátor reprezentace zachovávající typ výrazu Vstup (BYTE > BYTE, ...)

Barva - konstanta typu BYTE určující barvu vlastního sloupce

"V"|"H" - alternativní znaky určující typ sloupcového grafu jako vertikální nebo horizontální

Výška - konstanta typu BYTE (počet pixelů)

Šířka - konstanta typu BYTE (počet pixelů)

Pozadí	- konstanta typu BYTE určující barvu základní obdélníkové plochy
Min_Val	- minimální zobrazitelná hodnota - proměnná stejného typu jako vstup
Max_Val	- maximální zobrazitelná hodnota - proměnná stejného typu jako vstup
LoWar	- hodnota dolní varovné meze - proměnná stejného typu jako vstup
HiWar	- hodnota horní varovné meze - proměnná stejného typu jako vstup
LoAlm	- hodnota dolní havarijní meze - proměnná stejného typu jako vstup
HiAlm	- hodnota horní havarijní meze - proměnná stejného typu jako vstup
\MARK\	- odkaz na marker určující polohu prvního zobrazovaného znaku

## 5.7 \$TIGR - graf časových průběhů analogových signálů

### Určení reprezentace a její charakteristika

Standardní reprezentace \$TIGR je určena k zobrazování grafů časových průběhů nejvýše dvou proměnných do obdélníkového prostoru se společnou horizontálně orientovanou časovou osou. Posloupnosti vzorků, zpracované v \$TIGR do formy bodových grafů nebo grafů složených z navazujících úseček, mohou být této reprezentaci předány jako jednodimenzionální pole hodnot typu BYTE | FIXP | FLTP nebo může jít o posloupnosti získávané pomocí standardní funkce [REC](#) záznamem do cyklických zásobníků.

Výběr hodnot pro zobrazení může probíhat počínaje aktuálním indexem zásobníku [REC](#), nebo indexem "zmrazeným" funkcí REC v závislosti na specifikované události. V obou variantách může výběr historických hodnot začínat se zadaným zpožděním vůči zvolenému indexu. Posun zpracovávaného "časového okna" v delší posloupnosti vzorků je při případné změně momentální hodnoty parametru "zpoždění" prakticky okamžitý (v závislosti na periodě aktualizace skutečných hodnot v COLEDA). Posun okna může řídit operátor, nebo může být výsledkem zpracování dat - např. vyhodnocení lokálních extrémů v historických záznamech. Nejnovější hodnoty se projevují svislými souřadnicemi křivek na pravém okraji obdélníkové plochy vyhrazené pro všechny křivky určitého volání \$TIGR; hodnoty starší, resp. s vyšší hodnotou indexu vstupního pole, ovlivňují svislé souřadnice křivek s rostoucí vzdáleností od pravého okraje zobrazovací plochy. Zobrazované křivky jsou tedy postupně "odsouvány" směrem doleva, "nejstarší body" na levém okraji zobrazovací plochy mizí a na pravém okraji plochy jsou křivky doplňovány aktuálními body, popř. úsečkami.

Časová osa zobrazovaná na spodním okraji plochy křivek je rozdělena na pevně zadaný počet úseků s popisem začínajícím v místě první svislé osy nejvyšší hodnotou a směrem doleva s hodnotami postupně snižovanými o hodnotu parametru "krok\_popisu\_na\_ose\_x". Časová osa je dále popsána uživatelsky zadaným řetězcem časové jednotky (obvykle [s], [min] nebo [hod]).

Jednotlivé křivky mohou být barevně odlišeny - stejnými barvami jsou automaticky popisovány příslušné svisle orientované stupnice s popisem zadaného počtu úseků každé stupnice a se zadaným znakovým řetězcem fyzikální jednotky.

V prostoru popisu stupnice dané křivky je v číselné formě zobrazována také hodnota nejnovějšího vzorku příslušného signálu (vzorku pro nulový posun vůči aktuálnímu nebo zmrazenému indexu záznamu [REC](#), resp. pro první prvek zobrazeného pole). Aktuální řetězec tohoto číselného údaje je zobrazován v obdélníkové plošce, jejíž umístění ve svislém směru odpovídá souřadnici prvního zobrazovaného bodu dané křivky. Tato ploška může částečně nebo úplně překrýt popis svislé osy grafu.

Přepočet jednotlivých vstupních hodnot vzorku na svislé souřadnice křivky může probíhat pro každou křivku podle zadané specifikace reprezentace (obecně nelineárně) a dále (již lineárně) podle momentálních hodnot parametru "maximální\_zobrazitelná\_hodnota" a parametru "minimální\_zobrazitelná\_hodnota". Rozsah zobrazitelných hodnot lze tedy za chodu zásahem operátora nebo algoritmicky modifikovat - účinek změny měřítka zobrazení a popř. i nelineární charakteristiky specifické reprezentace je pro celou křivku jednorázový.

Vedle skupiny vstupů (polí nebo odkazů na záznamy REC) zpracovávaných do formy jednotlivých křivek, je možné zadat pro každou křivku nejvýše další čtyři vstupní parametry - signály, jejichž aktuální hodnoty určují vertikální souřadnice horizontálně vedených přímk, které mohou na ploše grafu vymezovat pásma povolených hodnot vstupních

veličin resp. meze výstražných a havarijních stavů pro jednotlivé křivky. Momentální hodnoty výstražných a havarijních mezí mohou být v číselné formě zobrazovány v prostoru stupnice příslušné křivky.

Každá křivka může být dále doplňována horizontálně vedenou osou, jejíž poloha může být - stejně jako u zobrazení mezí - fixní (určená konstantou tvořící příslušný skutečný parametr reprezentace), nebo proměnná (parametrem reprezentace je obecně výraz). Změna polohy osy nebo přímek mezí (Mez1, ..., Mez4) je při změně hodnoty příslušné proměnné prakticky okamžitá - je proto možné zásahem operátora posouvat příslušnou čáru a nastavením např. na bod staršího lokálního extrému může operátor odečíst jeho velikost.

Celková šířka plochy přepisované na obrazovce reprezentací TIGR je závislá jak na zadaném parametru "počet\_úseků\_na\_ose\_x", "počet\_vzorků\_v\_úseku" a parametru "počet\_pixelů\_na\_vzorek", tak i na počtu zobrazovaných křivek a tím i paralelně zobrazovaných svislých stupnic a také na délce popisu těchto stupnic, tj. délce řetězce "jednotka\_na\_ose\_y".

### Formát volání \$TIGR

Volání reprezentace \$TIGR má variabilní počet vstupních parametrů  $10 + N * 14$  - deset parametrů společných všem křivkám a čtrnáct parametrů příslušných pro N křivek:

```
REP Enable, Výška, BarvaPozadí, BarvaPopisu, PočetÚsekůNaOseX,  
KrokPopisuOsyX, ČasováJednotka, PočetVzorkůÚseku,  
Počet_pixelů_na_vzorek, index aktuální|zmrazený, zpoždění,  
/* následují parametry první křivky */  
Barva_grafu, Tloušťka_čáry, Zdroj, Reprezentace, MinVal, MaxVal,  
Počet_úseků_na_ose_y, Jednotka_na_ose_y, Osa, Meze_číslem,  
Mez1, Mez2, Mez3, Mez4  
/* zde by mohly následovat parametry druhé a případně i dalších křivek */  
> $TIGR > \MARK\;
```

### Parametry reprezentace

Enable	- proměnná typu BOOL určující, zda se má skupina grafů zobrazit
Výška 1000	- konstanta udávající výšku obdélníkové plochy pro TIGR v počtu pixelů v rozsahu 8 až 1000
BarvaPozadí	- barva pozadí obdélníkové plochy pro vykreslení křivek jako konstanta typu BYTE
BarvaPopisu BYTE	- barva pro zobrazení os, jejich popisu a horizontálních úseček mezí, jako konstanta typu BYTE
PočetÚsekůNaOseX BYTE	- počet stejných úseků na časové ose se samostatným popisem hranic úseků - konstanta BYTE
KrokPopisuOsyX	- konstanta typu BYTE určující rozdíl mezi hodnotami popisu hranic jednotlivých úseků na časové ose
Časová jednotka	- řetězcová konstanta maximální délky osm znaků určená pro popis časové osy
PočetVzorkůÚseku	- konstanta typu FIXP určující počet zpracovávaných vzorků (z REC nebo pole), jejichž hodnoty určují pro každou křivku její svislé souřadnice v rámci jednoho úseku
Počet_pixelů_na_vzorek	- konstanta typu BYTE určující délku úsečky spojující sousední vzorky (je-li = 1, je graf bodový)
Index_aktuální/zmrazený/	- znak "C" resp. "F" určující, zda jsou ze záznamu REC vybírány vzorky vztahené k aktuálnímu resp. zmrazenému indexu
Zpoždění	- proměnná typu FIXP, který určuje okamžitý posun časového okna zobrazených hodnot vůči zvolenému indexu v záznamu REC nebo ve vstupním poli, tedy hodnotu indexu prvního zpracovávaného vzorku

Parametry jednotlivých křivek:

Barva_grafu	- barva křivky jako konstanta typu BYTE
Tloušťka_čáry	- konstanta BYTE určující tloušťku čáry v počtu pixelů
Zdroj	- identifikátor proměnné určující vazbu na záznam REC nebo na vstupní pole
Reprezentace svislé souřadnice křivky	- identifikátor specifické reprezentace, která je aplikována na vstupující vzorek před určením
MinVal	- proměnná určující minimální zobrazitelnou hodnotu ze Zdroj
MaxVal	- proměnná určující maximální zobrazitelnou hodnotu ze Zdroj
Počet_úseků_na_ose_y	- konstanta BYTE určující počet úseků dělicích svislou osu na části shodné délky
Jednotka_na_ose_y	- řetězcová konstanta maximální délky 12ti znaků určená pro popis svislé osy
Osa	- výraz určující svislou souřadnici horizontálně vedené osy
Meze_číslem číselné formě	- příznak "A" nebo "N" určující, zda mají být momentální hodnoty mezi zobrazovány v
Mez1	- proměnná určující svislou souřadnici horizontálně vedené mezní čáry č. 1
Mez2, Mez3, Mez4	- obdoba Mez1 pro další 3 mezní čáry
\MARK\	- odkaz na marker určující polohu prvního zobrazovaného znaku

### Zpracování dat v \$TIGR

Hodnota proměnné Enable určuje, zda mají, či nemají v dané periodě aktualizace zobrazení proběhnout všechny dále uvedené výpočetní operace a vykreslení obdélníkového pozadí, os s popisem a vlastních křivek. Pro Enable == 0 zpracování reprezentace končí.

Pro každou křivku je určen koeficient K, udávající kolik pixelů odpovídá (ve svislém směru) změně hodnoty zobrazovaného vzorku o jednotkový přírůstek:

$$\text{Výška} / (\text{Max\_Val} - \text{Min\_Val}) > K,$$

Pro každý vzorek "Vstup" křivky se dále určí svislá souřadnice y v počtu pixelů jako posun zobrazovaného bodu vůči spodnímu okraji zobrazovací plochy, tj. vůči linii odpovídající hodnotě Min\_Val:

$$(\text{Vstup} - \text{Min\_Val}) * K > y,$$

Výsledná svislá souřadnice se zaokrouhlí a konvertuje na celé číslo a provede se kontrola - vyloučení hodnot y >> Vyska a y << 0, tedy hodnot ležících mimo momentálně platný rozsah <Min\_Val, Max\_Val>. Body, které by měly ležet mimo vyhrazenou plochu grafu jsou zobrazovány svislou úsečkou směřující příslušným směrem ven z plochy grafu.

Tento algoritmus se používá v případě, že nebyla pro danou křivku zadána specifická reprezentace (4. parametr křivky je prázdný). Je-li tato reprezentace zadána, předchází výpočtu souřadnice y transformace vstupního vzorku přes tuto reprezentaci:

$$\text{Vzorek } \$\text{Reprezentace} > \text{Vstup}$$

Reprezentace používané jako samostatný parametr křivek musí zachovávat typ zpracování proměnné, tzn. že pro "Vzorek" typu FIXP musí být i výstup reprezentace "Vstup" typu FIXP .... Stejná reprezentace je pro danou křivku automaticky používána i k předzpracování dalších parametrů - Min\_Val, Osa, Mez1, Mez2, Mez3, Mez4, to však předpokládá splnění dalšího požadavku na tuto reprezentaci, totiž její "obousměrnost", tedy existenci obou bloků (těl) reprezentace pro přímou i zpětnou transformaci.

## 5.8 \$TICUR - graf průběhů analog. signálů s údaji absolutního času

### Určení reprezentace a její charakteristika

Standardní reprezentace \$TICUR je určena k zobrazování grafů časových průběhů nejvýše dvou proměnných do obdélníkového prostoru se společnou horizontálně orientovanou časovou osou. Posloupnosti vzorků, zpracované v \$TICUR do formy bodových grafů nebo grafů složených z navazujících úseček, jsou této reprezentaci předávány v jednodimenzionálních polích hodnot typu BYTE | FIXP | FLTP. Časové údaje příslušející jednotlivým skupinám vzorků (se společnou časovou souřadnicí) jsou reprezentaci předávány ve dvojici dvojdimenzionálních polí pro datum a čas ve formátech \$DATE a \$TIME (každý okamžik je určen dvěma trojicemi hodnot typu BYTE), nebo alternativně v jednodimenzionálním poli typu LONG (údaj sekund vůči začátku r.1993). Skupina zobrazovaných signálů tedy nemusí být vzorkována periodicky - může jít o asynchronní záznamy událostí, které mohou pocházet z tzv. cyklických archivních binárních diskových souborů plněných standardní funkcí WRITEC nebo AKTC a do polí poskytovaných reprezentaci \$TICUR lze v tomto případě vzorky i údaje absolutního času plnit pomocí funkce READC.

Reprezentace TICUR umožňuje řídit prostřednictvím dvojice BOOL parametrů zobrazování grafu (včetně os a jejich popisu) a aktualizaci křivek podle momentálního obsahu vstupních polí - prostředky uživatelského programu lze zajistit zobrazování různých skupin křivek do jediné obdélníkové plochy (společný marker) s dynamickou volbou grafu operátorem, popř. i automaticky. Do stejného prostoru mohou alternativně zobrazovat další reprezentace, u nichž je aktualizace prováděna jen při změnách vstupních signálů ( \$GMAC nebo \$GTEXT), mohou tedy grafy např. dočasně překrýt GMACem zobrazené technologické schéma včetně dynamických grafických symbolů, různá menu a pod. (aplikační program musí po zrušení grafu vyvolat změnu indexů vstupujících do příslušných GMAC a GTEXT, přičemž naopak v periodě prvotního vyvolání TICUR je vhodné indexy násilně změnit na hodnotu, pro kterou zobrazují GMACy a GTEXTy např. jen plošky vybarvené neutrální barvou pozadí; volání TICURů by mělo být v programu níže).

Barva pozadí grafu, os, jejich popisu a jednotlivých křivek je rovněž - stejně jako rozsah zobrazovaných hodnot jednotlivých křivek - říditelná za chodu (prostřednictvím pole Paleta). Také styl zobrazení (izolované pixely nebo lomená čára) lze měnit za chodu.

Poloha (časový údaj počátku) resp. šířka časového okna odpovídajícího staticky vymezené ploše grafu je řízena vstupními parametry s formátem \$DATE a \$TIME resp. skupinou tří FIXP proměnných (počet dní, počet hodin a počet minut).

Popis vodorovné osy grafu není - zejména z prostorových důvodů - zajišťován samotnou reprezentací TICUR; osa může být popisována časovými údaji např. s využíváním reprezentace \$CD, jejímž parametrem mohou být reprezentace DATE a TIME.

Svislá osa může být zobrazována (včetně popisu mezními hodnotami) pro každou křivku nezávisle (na přání i dynamicky) a její popis lze řídit algoritmicky.

Vstupní hodnoty vzorků mohou být (pro každou křivku individuálně) před vlastním vykreslením zpracovány prostřednictvím zadané standardní reprezentace společně i pro hodnoty zobrazitelného minima a maxima. Jakákoli změna hodnoty některého minima či maxima vyvolá - stejně jako změna polohy nebo šířky časového okna - vyžádání vzorků od počátku okna a nové vykreslení všech křivek.

### Formát volání \$TICUR

Volání reprezentace \$TICUR má variabilní počet vstupních parametrů  $18 + N * 6$  - osmnáct parametrů společných všem křivkám a šest parametrů pro každou z N křivek:

```
REP Zobrazuj, Aktualizuj, Výška, Šířka, Paleta, ModZobraz,  
DatumOd, CasOd, DatumPosl, ČasPosl, NelzeZobrazit, AktualizPrvků,  
PočetDní, PočetHodin, PočetMinut, PoleDatum, PoleČas, PoleČasLong,  
/* parametry 1. křivky */  
PoleVstup, Reprezentace, Minimum, Maximum, PopisOsyYAno, PopisOsyY,  
/* příp. parametry 2. křivky */  
> $TICUR > \MARK\;
```

## Parametry reprezentace

Zobrazuj	- proměnná typu BOOL určující, zda se má skupina grafů zobrazit
Aktualizuj	- proměnná typu BOOL určující, zda se má skupina grafů aktualizovat
Výška	- konstanta udávající výšku obdélníkové plochy pro TICUR v počtu pixelů
Šířka	- konstanta udávající šířku obdélníkové plochy pro TICUR v počtu pixelů,
Paleta	- proměnná typu BYTE ARRAY udávající barvu pozadí, popisu, krivky1,..
ModZobraz	- proměnná typu BYTE udávající mód zobrazení - mód: 0 1 .. izolované pixely lomená čára
DatumOd	- proměnná typu BYTE ARRAY [#3] udávající počáteční datum pro zobrazení
CasOd	- proměnná typu BYTE ARRAY [#3] udávající počáteční čas pro zobrazení
DatumPosl	- proměnná typu BYTE ARRAY [#3] udávající datum posledního zobrazovaného vzorku
ČasPosl	- proměnná typu BYTE ARRAY [#3] udávající čas posledního zobrazovaného vzorku
NelzeZobrazit	- proměnná typu BOOL výstup - ve vstupech byl příliš nový vzorek
AktualizPrvků	- proměnná typu FIXP počet platných vstupních aktualizovaných vzorků
PočetDní	- proměnná typu FIXP šířka intervalu zobrazení - počet dnů
PočetHodin	- proměnná typu FIXP šířka intervalu zobrazení - počet hodin
PočetMinut	- proměnná typu FIXP šířka intervalu zobrazení - počet minut
PoleDatum	- proměnná typu BYTE ARRAY [#m][#3] data vzorků z PoleVstup
PoleČas	- proměnná typu BYTE ARRAY [#m][#3] čas vzorků z PoleVstup
PoleČasLong	- proměnná typu LONG ARRAY [#m] čas vzorků z PoleVstup v [s]

Parametry první křivky:

PoleVstup	- proměnná typu BYTE FIXP FLTP ARRAY [#m] - vzorky ke zpracování
Reprezentace	- Identifikátor reprezentace pro hodnoty 1. křivky
Minimum	- minimální zobrazitelná hodnota
Maximum	- maximální zobrazitelná hodnota
PopisOsYAno	- znak "Y"   "N" - osu Y zobrazuj nezobrazuj
PopisOsY	- proměnná typu CHAR ARRAY #n - řetězcová proměnná (konst.) pro popis osy Y
\MARK\	- odkaz na marker určující polohu prvního zobrazovaného znaku

Přičemž #m označuje počet prvků vstupních polí časových údajů a polí PoleVstup vlastních vzorků pro jednotlivé křivky - rozsah těchto polí nemusí být nutně shodný, TICUR zpracovává vždy jen počet prvků určený vstupem AktualizPrvků, který by měl odpovídat počtu platných údajů. TICUR rozsahy polí podle jejich deklarace nezpracovává. Popis osy y může mít maximálně 8 znaků. PoleČasLong je prozatím rezerva (nezadává se!) určená pro budoucí usnadnění vazby na funkci READC.

## Zpracování dat v reprezentaci \$TICUR

V rozsahu zadaného časového okna (určeného počátkem a šířkou intervalu) prohledává TICUR při vyvolání na obrazovku určený počet aktualizovaných prvků vstupních polí časových údajů (počet aktualizovaných prvků může poskytnout funkce READC), nalezené vzorky zpracovává a na závěr naplní hodnoty celkem šesti prvků dvou výstupních polí časovým údajem poslední ("nejmladší") zpracované skupiny vzorků.

Při následující aktualizaci grafu hledá TICUR ve vstupních polích pouze vzorky chybějící k doplnění křivek do konce zadaného časového okna a plní opět výstupní pole čas.údaje posledních zpracovávaných vzorků. Dokud nejsou změněny údaje požadovaného počátku a šířky časového okna, jsou při příchodu nových vzorků křivky pouze dokreslovány na svých pravých koncích. Uživatel tedy má možnost měnit polohu časového okna pro TICUR jen v delších periodách a zabránit tak relativně časově náročnému překreslování celých křivek při každé aktualizaci.

Jestliže se ve vstupních polích objeví alespoň jeden časový údaj ležící vně zadaného časového okna ve směru narůstajícího času, nastaví TICUR výstup typu BOOL (s významem "Nelze zobrazit") - tento signál lze interpretovat např. jako žádost o posun časového okna směrem k novějším vzorkům (doprava), a to obvykle o pevně zvolený posun (přičtení k předcházejícímu čas.údaji počátku okna, včetně "přenosů do vyšších řádů", musí zajistit aplikační program).

Výstupy z TICUR mohou být zavedeny jako vstupy přímo do funkce READC (datum a čas Po), nebo mohou být vysílány i do vzdálených stanic v rámci žádosti o archivovaná data. Funkci READC se často požadavek na čas původu nejnovějších vzorků (datum a čas Do) předává přímo jako aktuální systémový čas (SDATE a STIME).

## 5.9 \$XYCUR - zobrazení grafů v souřadnicích x,y

### Určení reprezentace a její charakteristika

Standardní reprezentace \$XYCUR je určena k zobrazování závislostí dvojic veličin typu BYTE|FIXP|FLTP ve formě nejvýše dvou barevně odlišených množin bodů nebo alternativně lomených čar. Souřadnice x,y jednotlivých zobrazovaných bodů jsou reprezentaci XYCUR předávány v jednodimenzionálních polích.

Reprezentace XYCUR umožňuje dynamicky řídit vlastní zobrazování a aktualizace křivek prostřednictvím dvojice BOOL vstupů (Zobrazuj a Aktualizuj) a může sdílet společný zobrazovací prostor s voláními dalších reprezentací \$XYCUR, reprezentací \$TICUR, \$GMAC a \$GTEXT - aplikační program musí zajišťovat vzájemné vyloučení signálů Zobrazuj a změn indexů pro reprezentace \$GMAC a \$GTEXT.

### Formát volání \$XYCUR

Volání reprezentace \$XYCUR má variabilní počet vstupních parametrů  $6 + N * 13$  - šest parametrů společných všem křivkám a třináct parametrů pro každou z N křivek:

```
REP Zobrazuj, Aktualizuj, Výška, Šířka, Paleta, ModZobraz,
/* parametry 1.křivky */
PoleX, Reprezentace, MinimumX, MaximumX,
PoleY, Reprezentace, MinimumY, MaximumY,
IndexMIN, IndexMAX, PopisOsAno, PopisOsYX, PopisOsY,
/* příp. parametry 2.křivky */
> $XYCUR > \MARK\;
```

### Parametry reprezentace

Zobrazuj	- proměnná typu BOOL určující, zda se má skupina grafů zobrazit
Aktualizuj	- proměnná typu BOOL určující, zda se má skupina grafů aktualizovat
Výška	- konstanta udávající výšku obdélníkové plochy pro TICUR v počtu pixelů
Šířka	- konstanta udávající šířku obdélníkové plochy pro TICUR v počtu pixelů,
Paleta	- proměnná typu BYTE ARRAY udávající barvu pozadí, popisu, křivky1,..
ModZobraz	- proměnná typu BYTE udávající mod zobrazení - mod: 0 1 .. izolované pixely lomená čára

Parametry první křivky:

PoleX	- proměnná typu BYTE FIXP FLTP ARRAY [#m] - vzorky ke zpracování
Reprezentace	- Identifikátor reprezentace pro hodnoty osy X
MinimumX	- minimální zobrazitelná hodnota
MaximumX	- maximální zobrazitelná hodnota
PoleY	- proměnná typu BYTE FIXP FLTP ARRAY [#m] - vzorky ke zpracování

Reprezentace	- Identifikátor reprezentace pro hodnoty osy Y
MinimumY	- minimální zobrazitelná hodnota
MaximumY	- maximální zobrazitelná hodnota
IndexMIN	- proměnná typu FIXP - minimální index v polích PoleX, PoleY
IndexMAX	- proměnná typu FIXP - maximální index v polích PoleX, PoleY
PopisOsyYAno	- znak "Y"   "N" - osu Y zobrazuj nezobrazuj
PopisOsyX	- proměnná typu CHAR ARRAY #8 - řetězcová proměnná (konst.) pro popis osy X
PopisOsyY	- proměnná typu CHAR ARRAY #8 - řetězcová proměnná (konst.) pro popis osy Y
\MARK\	- odkaz na marker určující polohu prvního zobrazovaného znaku

Příčemž #m označuje počet prvků vstupních polí PoleX, PoleY vzorků pro jednotlivé křivky - rozsah těchto polí nemusí být nutně shodný, XYCUR zpracovává vždy jen počet prvků určený vstupy IndexMIN a IndexMAX, které by neměly vybočovat z oblasti platných údajů vstupních polí.

### Zpracování dat v reprezentaci \$XYCUR

Každá křivka se vykresluje samostatně na základě dvojic stejnohlých hodnot v polích PoleX, PoleY, a to počínaje indexem IndexMIN až do IndexMAX. Prvky vstupních polí mohou být zpracovány prostřednictvím reprezentace (obecně různé pro souřadnice x a y); výsledek je přepočítán s uvážením rozměrů celé plochy grafu a minimální a maximální zobrazitelné hodnoty v příslušném směru. Body, jejichž souřadnice padnou mimo rozsah vyhrazené plochy nejsou samozřejmě vykresleny, body následující (pro vyšší indexy) mohou být opět zobrazeny.

Je-li v barevné paletě zadáno číslo 255, XYCUR pozadí a osy s popisy vůbec nezobrazuje; toho lze využít pro vykreslování většího počtu křivek do společné obdélníkové plochy - několik volání XYCUR se odvolává na jediný marker a jen poslední z nich zajistí podbarvení plochy a vykreslení os s popisem.

Aktualizace zobrazení křivek může být řízena aplikačním programem dvěma různými způsoby. Jestliže je zadán parametr Aktualizuj, vyvolává jednotková hodnota zadané vstupní BOOL proměnné nové podbarvení celé plochy grafu a vykreslení jedné, popř. obou křivek podle hodnot vstupních polí určených rozsahem indexů IndexMIN, IndexMAX. Při nulovém vstupu Aktualizuj se dříve zobrazené křivky nepřemazávají novým podbarvením plochy grafu, ale při jakékoli změně vstupů IndexMIN, IndexMAX se takto určená série hodnot zpracuje a vykreslí.

Pokud není zadán parametr Aktualizuj, je aktualizace vyvolávána výhradně změnami hodnot vstupů IndexMIN, IndexMAX; navíc při vynulování hodnoty IndexMAX (alespoň pro jednu křivku) dojde k novému podbarvení plochy grafu.

Z uvedených vlastností XYCUR vyplývá, že tato reprezentace poskytuje programátorovi možnosti vytváření různých "efektů" nerealizovatelných pomocí reprezentací \$TICUR resp. \$TIGR. Ve společných souřadnicích lze zobrazovat větší počet křivek, které nemusí být grafy funkcí proměnné x nebo y. Dříve vykreslené křivky lze zachovávat a nové křivky popsane aktualizovanými hodnotami těchže polí mohou být vykreslovány v odlišné barvě.

## 5.10 \$TAB - zobrazení tabulky hodnot

### Určení reprezentace a její charakteristika

Standardní reprezentace \$TAB je určena pro zobrazování skupin hodnot získávaných ze záznamu REC (časové řady vzorků) nebo ze statického jednodimenzionálního nebo dvoudimenzionálního pole ve formě tabulky číselných údajů, resp. obecně tabulky řetězců. Konstantními parametry reprezentace \$TAB lze určit počet řádků a počet sloupců zobrazované tabulky, barvu pozadí (obdélníkové plochy tabulky) a barvu zobrazovaných čísel. Dále lze, obdobně jako u reprezentace \$TIGR, zadat identifikátor specifické reprezentace, určující transformaci, které jsou podrobeny všechny zpracovávané vzorky před zobrazením řetězců číselných údajů, avšak s tím rozdílem proti \$TIGR, že tato reprezentace musí mít výstupní typ CHAR a může jít o reprezentaci s jediným "směrem transformace". Formát zobrazení nelze určit přímo parametry volání \$TAB - je určen nepřímo - rovněž touto reprezentací.

Výběr zpracovávaných vzorků probíhá podobně jako u \$TIGR, tj. se zadaným posunem (zpožděním), a to vůči aktuálnímu resp. zmrazenému indexu ve standardní funkci REC. Nejstarší hodnoty z REC jsou v zobrazené tabulce

umístěny v jejím levém horním rohu (mají nižší index řádku i sloupce), hodnoty novější pak v řádcích směrem zleva doprava a dále v níže umístěných řádcích. Pro zobrazování prvků pole platí rovněž obvyklá konvence - sloupcový index j narůstá pro rostoucí adresu prvku pole v paměti rychleji než řádkový index i.

### Formát volání \$TAB

Volání reprezentace \$TAB má pevný formát:

```
REP Pozadí, Řádků, Sloupců, Index_aktuální|zmrazený, Zpoždění,  
Barva_čísel, Zdroj, Reprezentace > $TAB > \MARK\;
```

### Parametry reprezentace

Pozadí	- konstanta typu BYTE určující barvu pozadí
Řádků	- konstanta typu BYTE určující počet řádků tabulky
Sloupců	- konstanta typu BYTE určující počet sloupců tabulky
Index_aktuální zmrazený	- znak "C" resp. "F" určující zda jsou ze záznamu REC vybírány vzorky vztažené k aktuálnímu resp. zmrazenému indexu
Zpoždění	- výraz typu FIXP, který určuje okamžitý posun časového okna zobrazovaných hodnot vůči vybranému indexu záznamu REC nebo počátku vstupního pole
Barva_čísel	- konstanta typu BYTE
Zdroj	- identifikátor proměnné použité v některém volání REC (příp. i ve více voláních REC) nebo identifikátor vstupního pole
Reprezentace	- identifikátor reprezentace která určuje způsob zpracování všech hodnot, tj. obecně nelineární transformaci těchto hodnot a převod na řetězec se specifikovaným formátem čísla, popř. na libovolnou řetězcovou konstantu
\MARK\	- odkaz na marker určující polohu prvního zobrazovaného znaku

## 5.11 \$TEXT - zobrazení alternativních textových řetězců

### Určení reprezentace a její charakteristika

Reprezentace \$TEXT je určena pro zobrazování stavu určené BOOL proměnné (výrazu) ve formě různých textových řetězců se zadanou barvou a s určeným typem písma.

### Formát volání \$TEXT

Reprezentace \$TEXT má tento pevný formát volání:

```
REP Vstup, Řetězec1, B1, P1, Řetězec0, B0, P0, Typ > $TEXT > \MARK\;
```

### Parametry reprezentace

Vstup	- Výraz typu BOOL, který určuje k zobrazení Řetězec1 pro stav 1, nebo Řetězec0 pro stav 0.
Řetězec1,Řetězec0	- řetězcová proměnná nebo konstanta.
B1,B0	- Konstanta typu BYTE určující barvu zobrazovaných znaků příslušného řetězce.
P1,P0	- Konstanta typu BYTE určující barvu pozadí zobrazovaných znaků příslušného řetězce.
Typ	- Konstanta typu BYTE určující znakovou sadu pro všechny zobrazované znaky (prozatím pouze 0 - standardní sada).
\MARK\	- Odkaz na marker určující polohu prvního zobrazovaného znaku.

## 5.12 \$MTEXT - reprezentace skupiny proměnných textovými řetězci

### Určení reprezentace a její charakteristika

Reprezentace \$MTEXT je určena pro zobrazování stavu určené skupiny BOOL proměnných ve formě různých textových řetězců se zadanou barvou a s určeným typem písma.

### Formát volání \$MTEXT

Reprezentace \$MTEXT má počet vstupních parametrů závislý na počtu řídicích vstupů. Pro 2 vstupy je třeba určit 4 různé řetězce a jejich barvy, pro 3 vstupy 8 různých řetězců a jejich 8 barev,... Společným parametrem je TypPísma.

```
REP VstupN, VstupN-1, VstupN-2, ..., Vstup0,  
ŘetězecM, BarvaM, ŘetězecM-1, BarvaM-1, ..., Řetězec0, Barva0,  
TypPísma > $MTEXT > \MARK\;
```

### Parametry reprezentace

VstupN, VstupN-1, ..., Vstup0 - proměnné typu BOOL,  
ŘetězecM, ŘetězecM-1, ..., Řetězec0 - řetězcové proměnné nebo konstanty,  
BarvaM, BarvaM-1, ..., Barva0 - konstanty typu BYTE

### Zpracování dat v reprezentaci \$MTEXT

Po vyhodnocení všech vstupních výrazů typu BOOL je z jejich hodnot sestaveno binární číslo, a to tak, že Vstup0 určí 0-tý řád tohoto čísla, ... a VstupN určí jeho N-tý řád. Získané binární číslo určuje pořadí aktuální dvojice Řetězec, Barva určené k zobrazení. Jsou-li všechny BOOL vstupy nulové, je aktuální dvojice Řetězec0, Barva0. Pro všechny vstupy ve stavu 1 je aktuální dvojice odpovídající nejvyššímu binárnímu číslu, tj.: ŘetězecM, BarvaM.

## 5.13 \$GTEXT - zobrazení řetězce jako prvku pole řetězců

### Určení reprezentace a její charakteristika

Reprezentace \$GTEXT je určena pro zobrazování textových řetězců vybraných z pole řetězců podle indexu určeného momentální hodnotou řídicí proměnné typu BYTE nebo FIXP. Tentýž index určuje barvu zobrazovaných znaků a barvu pozadí odkazem do pole typu BYTE tvořícího tzv. paletu barev.

### Formát volání \$GTEXT

Reprezentace \$GTEXT má 4 vstupní parametry a jediný výstup:

```
REP Index1, PoleŘ, Index2, Paleta > $GTEXT > \MARK\;
```

### Parametry reprezentace

Index1 - proměnná typu BYTE nebo FIXP - index do pole řetězců  
PoleŘ - jednorozměrné pole řetězců (STAT CHAR #n ARRAY PoleŘ[i..j]);  
Index2 - proměnná typu BYTE nebo FIXP - index do palety barev  
Paleta - pole typu BYTE obsahující dvojice kódů barev: text-pozadí  
\MARK\ - odkaz na marker určující polohu prvního zobrazovaného znaku  
Parametry Index1 a Index2 musí být stejného typu.

### Zpracování dat v reprezentaci \$GTEXT

Index1 určí jeden z řetězců pole PoleŘ s maximální délkou n určenou v deklaraci tohoto pole a se skutečnou délkou nastavenou při inicializaci daného prvku řetězcovou konstantou, resp. komunikačním přiřazením, nebo exekucním

přiřazením řetězce do PoleŘ. Vybraný řetězec se zobrazí na barevném pozadí a s barvou určenou momentální hodnotou Index2-té dvojice prvků pole Paleta. Pokud hodnota proměnné Index1 vybočí z rozsahu pole PoleŘ, zobrazí se náhodný obsah paměti, avšak s délkou řetězce omezenou vždy nejvýše na n; barva textu a pozadí je rovněž v tomto případě náhodná.

Pokud se hodnota proměnné Index1 ani Index2 od minulé aktualizace nezměnila, neprovádí tato reprezentace nové zobrazení! Tato vlastnost odstraňuje nežádoucí "blikání" obrazu, šetří čas, ale umožňuje také "umělým" zafixováním indexů vyloučit dočasně aktualizaci příslušné části obrazovky a tím umožnit zobrazování jinými reprezentacemi do stejné oblasti obrazovky.

## 5.14 \$MAC - zobrazení alternativních makrosymbolů

### Určení reprezentace a její charakteristika

Reprezentace \$MAC je určena pro zobrazování stavu určené BOOL proměnné (výrazu) ve formě různých grafických makrosymbolů připravených dříve prostředky programového komplexu GELEDA. Alternativní makrosymbole se mohou lišit jak tvarově, tak i velikostí a barvami jednotlivých použitých grafických prvků.

### Formát volání \$MAC

Reprezentace \$MAC má tento pevný formát volání:

```
REP Vstup, Řetězec1, Řetězec0 > $MAC > \MARK\;
```

### Parametry reprezentace

Vstup - Výraz typu BOOL, který určuje k zobrazení makrosymbol určený obsahem konstanty Řetězec1 pro stav 1, nebo obsahem Řetězec0 pro stav 0.

Řetězec1,Řetězec0 - Řetězcové konstanty tvořené jmény diskových souborů (bez extenze, ale popř.také s "cestou") obsahujících grafické obrazy získané "exportem" z grafického editoru GELEDA.

\MARK\ - Odkaz na marker určující polohu prvního zobrazovaného znaku.

## 5.15 \$MMAC - reprezentace skupiny proměnných graf. makrosymbole

### Určení reprezentace a její charakteristika

Reprezentace \$MMAC je určena pro zobrazování stavu určené skupiny BOOL proměnných ve formě různých grafických makrosymbolů "vyexportovaných" z programového komplexu GELEDA.

### Formát volání \$MMAC

Reprezentace \$MMAC má počet vstupních parametrů závislý na počtu řídicích vstupů. Pro 2 vstupy je třeba určit 4 různé makrosymbole, pro 3 vstupy 8 různých makrosymbolů, ... .

```
REP VstupN, VstupN-1, VstupN-2, ..., Vstup0,  
ŘetězecM, ŘetězecM-1, ..., Řetězec0 > $MMAC > \M\;
```

### Parametry reprezentace

VstupN, VstupN-1, ...,Vstup0 - proměnné typu BOOL,

ŘetězecM, ŘetězecM-1, ...,Řetězec0 - řetězcové konstanty,

### Zpracování dat v reprezentaci \$MMAC

Po vyhodnocení všech vstupních výrazů typu BOOL je z jejich hodnot sestaveno binární číslo, a to tak, že Vstup0 určí 0-tý řád tohoto čísla, ... a VstupN určí jeho N-tý řád. Získané binární číslo určuje pořadí i aktuální řetězcové konstanty Řetězci. Jsou-li všechny BOOL vstupy nulové, je aktuální Řetězec0. Pro všechny vstupy ve stavu 1 je

aktuální řetězec, který odpovídá nejvyššímu binárnímu číslu, tj.: ŘetězecM. Řetězec je interpretován jako jméno souboru, který obsahuje grafický obraz získaný exportem z GELEDA. Vybraný obraz je vykreslen na místo určené markerem.

## 5.16 \$GMAC - zobrazení graf. makrosymbolu určeného indexem

### Určení reprezentace a její charakteristika

Reprezentace \$GMAC je určena pro zobrazování různých grafických makrosymbolů "vyexportovaných" z programového komplexu GELEDA a vybíraných na základě momentální hodnoty řídicí proměnné tvořící index do pole řetězců jmen diskových souborů.

### Formát volání \$GMAC

Reprezentace \$GMAC má dva vstupní parametry a jediný výstupní parametr:

```
REP Index, PoleŘ > $GMAC > \MARK\;
```

### Parametry reprezentace

Index - proměnná typu BYTE

PoleŘ - pole řetězců

### Zpracování dat v reprezentaci \$GMAC

Hodnota proměnné Index určí řetězec z PoleŘ. Řetězec je dále interpretován jako jméno souboru (příp.i s posloupností jmen adresářů tvořících "cestu"), který obsahuje grafický obraz získaný exportem z GELEDA. Vybraný obraz je vykreslen na místo určené markerem. PoleŘ je nutné předem naplnit řetězci popisujícími existující diskové soubory obsahující makrosymbole (vlastně obrazy) - exekucním přiřazením nebo komunikačně; nejčastěji inicializací řetězcovými konstantami.

Pokud se hodnota proměnné Index od minulé aktualizace nezměnila, neprovádí tato reprezentace nové zobrazení! Tato vlastnost odstraňuje nežádoucí "blikání" obrazu, šetří čas, ale umožňuje také "umělým" zafixováním indexu vyloučit dočasně aktualizaci příslušné části obrazovky a tím umožnit zobrazování jinými reprezentacemi do stejné oblasti obrazovky.

## 5.17 \$READ - zobrazení grafického obrazu

### Určení reprezentace a její charakteristika

Standardní reprezentace \$READ je určena pro zobrazování technologických schémat nebo jiných grafických obrazů připravených pomocí grafického editoru GELEDA.

### Formát volání \$READ

Reprezentace \$READ má jediný vstupní parametr, kterým je řetězcová konstanta určující diskový soubor (včetně cesty) exportovaného obrazu. Jediným výstupním parametrem je odkaz určující polohu levého horního rohu obdélníkové plochy obrazu.

```
Řetězec > $READ > \MARK\;
```

### Zpracování dat v reprezentaci \$READ

Je-li v režimu Run komplexu COLEDA dosaženo koincidence levého horního rohu výřezu zdrojového textu s polohou určenou odkazem tvořícím výstupní parametr volání reprezentace \$READ, je obraz určený vstupním parametrem tohoto volání zobrazen, a to dříve, než proběhne zobrazení všech ostatních reprezentací s grafickým či alfanumerickým výstupem.

## 6. Volání standardních funkcí a reprezentací - přehled formátů

Poznámka

Všude, kde smí být použita proměnná, může být použit skalární parametr nebo (u vstupních parametrů) výraz - výjimkou z tohoto pravidla je tzv. zdrojová proměnná funkce REC.

### Funkce přenosu dat

OUTR > Číslo kanálu  
konstanta 0/1

Vstup1, Reprez1, Koment1, ..., KomentN > OUTF > Soubor  
Prom. Ident. ř.konst. ř.konst. ř.konst.

Proměň1, Reprez1, ..., ProměňN, ReprezN > OUTS > PoleZnaků  
Prom. Ident. Prom. Ident. ř.prom.

ZprávaVysílaná, ČísloBUSpříjemce > PUTARR > ÚspěchVyslání  
STAT BYTE ARRAY ř.konst. BOOL

ČísloBUSvysílače > GETARR > ZprávaPřijatá, ÚspěchPřijetí  
ř.konst. STAT BYTE ARRAY BOOL

Volání funkcí WRITEC, READC, AKTC a LASTTC pouze v kapitole 4.

### Funkce pro ovládání akustického výstupu

Start > BEEP;  
BOOL

Start, Melody > AUDIO;  
BOOL FIXP ARRAY

### Vstupní a výstupní zpracování signálů

Input, Typ\_IPR > IPR > UnifI  
Prom. Č.konst. Proměnná  
BYTE (IL) BYTE | FIXP

Výstup, Typ\_OPR > OPR > Výstupní signál  
Prom. Č.konst. Proměnná  
FIXP BYTE (OL)

## Zpracování signálů - dynamické funkce

Není-li typ proměnné uveden, je FIXP (pro typy BYTE, FLTP, ale i pro BOOL probíhají automaticky konverze); vstupní parametry mohou být tvořeny výrazy.

-----  
SP, PV, Aux, AUT, FOLL.SP, GAIN, Ti, Td, Kaux, Tau,  
          BOOL                  FLTP FLTP FLTP FLTP FLTP  
LoLimDY, HiLimDY, LoLimY, HiLimY > PID > Y  
-----

-----  
SP, PV, GAIN, Td, Tau, LoLimY, HiLimY > PDC > Y  
          FLTP FLTP FLTP  
-----

U, TAMin, Tmin > PIM > Y1, Y2  
                          BOOL BOOL

U, Bi, Yi > SUM > Y  
          FLTP BOOL FLTP          FLTP

U > DIF > Y

U, Interval > MMA > Y, Index  
          FLTP                          FLTP

U, Interval > MMI > Y, Index  
          FLTP                          FLTP

U, Interval > MAV > Y  
          FLTP                          FLTP

U, Delay > TDL > Y  
          FLTP                          FLTP

U, Tau > FIL > Y  
          FLTP

U > SRG > Pole  
          BYTE|FIXP|FLTP          BYTE|FIXP|FLTP

Start,Zdroj.prom.,N > REC

BOOL

U, LoLimU, HiLimU, Hyst > CMP > Low

BOOL

nebo

U, LoLimU, HiLimU, Hyst, > CMP > Out, High

BOOL BOOL

### Zpracování signálů - statické funkce

B, U1, U2 > SEL > Y

BOOL

U, X1, Y1, ... , Xn, Yn > NLT > Y

FLTP/FIXP

FLTP/FIXP

U, LoLimY, HiLimY > SAT > Y

FLTP/FIXP

FLTP/FIXP

U, LoLimU, HiLimU > DBN > Y

FLTP/FIXP

FLTP/FIXP

U1, U2, ... > MAX > Y

FLTP/FIXP

FLTP/FIXP

U1, U2, ... > MIN > Y

FLTP/FIXP

FLTP/FIXP

Vstup > SQT > Y

Vstup > ABS > Y

PoleChar,Reprez,MaxPocZnaku > ATN > Chyba, Y;

CHAR

BYTE

BOOL

BYTE/FIXP/FLTP

### Zpracování dvouhodnotových signálů

(není-li uvedeno jinak, jsou parametry výrazy typu BOOL)

B, U1, U2 > SEL > X

U > SET > X

U > RES > X

U, Delay > DSR > X

*FIXP*

U, Delay > SDR > X

*FIXP*

U, Delay > MSF > X

*FIXP*

U > IMP > X

U > NIMP > X

U > DIF > X

CU, CD, Bi, Yi > CNT > X

*FIXP* *FIXP*

Bn, ..., B2, B1, B0 > CPS > Vystup

*BYTE* / *FIXP*

Vstup > DCPS > Bn, ..., B2, B1, B0

*FIXP*

### Formáty volání standardních reprezentací se vstupem z markerů

REP Proměnná < < \MARK\;	/* bez přepočtu (do 255 pro BYTE, do 32767 pro FIXP)*/
REP Proměnná < <u>FP\$</u> < \MARK\;	/* v % rozsahu FIXP proměnné (8000) */
REP Proměnná < <u>F1K\$</u> < \MARK\;	/* průtok 0 až 1000.0 m3/h */
REP Proměnná < <u>FK4\$</u> < \MARK\;	/* průtok 0 až 400.0 m3/h */
REP Proměnná < <u>FK25\$</u> < \MARK\;	/* průtok 0 až 250.0 m3/h */
REP Proměnná < <u>FK16\$</u> < \MARK\;	/* průtok 0 až 160.0 m3/h */
REP Proměnná < <u>FF\$</u> < \MARK\;	/* průtok -999.9 až 999.9 m3/h */
REP Proměnná < <u>MM\$</u> < \MARK\;	/* průtok -999.9 až 999.9 t/h */
REP Proměnná < <u>P4M\$</u> < \MARK\;	/* tlak -4.000 až 4.000 MPa */
REP Proměnná < <u>P1M6\$</u> < \MARK\;	/* tlak -1.600 až 1.600 MPa */

REP Proměnná < PP\$ < \MARK\; /\* tlak -9.999 až 9.999 MPa \*/  
 REP Proměnná < T2\$ < \MARK\; /\* teplota -050.0 až 200.0 °C \*/  
 REP Proměnná < T5\$ < \MARK\; /\* teplota 000.0 až 500.0 °C \*/  
 REP Proměnná < TT\$ < \MARK\; /\* teplota -999.9 až 999.9 °C \*/  
 REP Proměnná < EE\$ < \MARK\; /\* energie -999.9 až 999.9 GJ \*/  
 REP Proměnná < WW\$ < \MARK\; /\* výkon -99.99 až 99.99 MW \*/  
 REP Proměnná < LL\$ < \MARK\; /\* hladina -999.9 až 999.9 cm \*/  
 REP Proměnná < QQ\$ < \MARK\; /\* objem -999.9 až 999.9 m3 \*/  
 REP Proměnná < GG\$ < \MARK\; /\* hmotnost -999.9 až 999.9 t \*/

### Formáty volání standardních reprezentací s výstupem na markery

REP Proměnná > > \MARK\; /\* zobrazení v bitech (do 32767)\*/  
 REP Proměnná > \$ > \MARK\; /\* zobrazení v % rozsahu (8000) \*/  
 REP Proměnná > \$F1K > \MARK\; /\* průtok 0 až 1000.0 m3/h \*/  
 REP Proměnná > \$FK4 > \MARK\; /\* průtok 0 až 400.0 m3/h \*/  
 REP Proměnná > \$FK25 > \MARK\; /\* průtok 0 až 250.0 m3/h \*/  
 REP Proměnná > \$FK16 > \MARK\; /\* průtok 0 až 160.0 m3/h \*/  
 REP Proměnná > \$FF > \MARK\; /\* průtok -999.9 až 999.9 m3/h \*/  
 REP Proměnná > \$MM > \MARK\; /\* průtok -999.9 až 999.9 t/h \*/  
 REP Proměnná > \$P4M > \MARK\; /\* tlak -4.000 až 4.000 MPa \*/  
 REP Proměnná > \$P1M6 > \MARK\; /\* tlak -1.600 až 1.600 MPa \*/  
 REP Proměnná > \$PP > \MARK\; /\* tlak -9.999 až 9.999 MPa \*/  
 REP Proměnná > \$T2 > \MARK\; /\* teplota -050.0 až 200.0 °C \*/  
 REP Proměnná > \$T5 > \MARK\; /\* teplota 000.0 až 500.0 °C \*/  
 REP Proměnná > \$TT > \MARK\; /\* teplota -999.9 až 999.9 °C \*/  
 REP Proměnná > \$EE > \MARK\; /\* energie -999.9 až 999.9 GJ \*/  
 REP Proměnná > \$WW > \MARK\; /\* výkon -99.99 až 99.99 MW \*/  
 REP Proměnná > \$LL > \MARK\; /\* hladina -999.9 až 999.9 cm \*/  
 REP Proměnná > \$QQ > \MARK\; /\* objem -999.9 až 999.9 m3 \*/  
 REP Proměnná > \$GG > \MARK\; /\* hmotnost -999.9 až 999.9 t \*/  
  
 REP Soubor > \$READ > \MARK\; /\* přenos grafického obrazu ze souboru na obrazovku \*/  
 Ř.konst

REP Vstup, Reprez., BarvaPísma, BarvaPozadí, Typ > \$D > \MARK\;

REP Vstup, Reprezentace, IndexBarvy, Paleta, Typ > \$CD > \MARK\;

```

REP Vstup, Reprezentace, Barva, "V"|"H", Výška, Šířka, Pozadí,
  Min_Val, Max_Val, LoWar, HiWar, LoAlm, HiAlm > $BAR > \MARK\;

REP Pozadí, Řádků, Sloupců, Index_aktuální|zmrazený, Zpoždění,
  Barva_čísels, Zdroj, Reprezentace > $TAB > \MARK\;

REP Výška, Barva_pozadí, Barva_popisu, Počet_úseků_na_ose_x,
  Krok_popisu_osy_x, Časová_jednotka, Počet_vzorků_v_úseku,
  Počet_pixelů_na_vzorek, index_aktuální|zmrazený, zpoždění,
  /* následují parametry jednotlivých křivek */
  Barva_grafu, Tloušťka_čáry, Zdroj, Reprezentace, MinVal, MaxVal,
  Počet_úseků_na_ose_y, Jednotka_na_ose_y, Osa, Meze_číslem,
  Mez1, Mez2, Mez3, Mez4
  /* zde by mohly následovat parametry druhé a případně i dalších křivek */
  > $TIGR > \MARK\;

REP Zobrazuj, Aktualizuj, Výška, Šířka, Paleta, ModZobraz,
  DatumOd, CasOd, DatumPosl, CasPosl, NelzeZobrazit, AktualizPrvků,
  PočetDní, PočetHodin, PočetMinut, PoleDatum, PoleČas, PoleČasLong,
  /* parametry 1. křivky */
  PoleVstup, Reprezentace, Minimum, Maximum, PopisOsYAno, PopisOsY,
  /* příp. parametry 2. křivky */
  > $TICUR > \MARK\;

REP Zobrazuj, Aktualizuj, Výška, Šířka, Paleta, ModZobraz,
  /* parametry 1.křivky */
  PoleX, Reprezentace, MinimumX, MaximumX,
  PoleY, Reprezentace, MinimumY, MaximumY,
  IndexMIN, IndexMAX, PopisOsAno, PopisOsYX, PopisOsY,
  /* příp. parametry 2.křivky */
  > $XYCUR > \MARK\;

REP Vstup, Řetězec1, Barva1, Řetězec0, Barva0, TypPísma > $TEXT > \MARK\;

REP VstupN, VstupN-1, VstupN-2, ..., Vstup0,
  ŘetězecM, BarvaM, ŘetězecM-1, BarvaM-1, ..., Řetězec0, Barva0,
  TypPísma > $MTEXT > \MARK\;

REP Index1, PoleŘ, Index2, Paleta > $GTEXT > \MARK\;

REP Vstup, Řetězec1, Řetězec0 > $MAC > \MARK\;

```

REP VstupN, VstupN-1, VstupN-2, ..., Vstup0,  
ŘetězecM, ..., Řetězec0 > \$MMAC > \MARK\;

REP Index, PoleŘ > \$GMAC > \MARK\;

Kódy funkčních kláves, které jsou nejčastěji používány pro volbu výřezu pro zobrazení hodnot

	Klávesa	SHIFT + klávesa	CTRL + klávesa	ALT + klávesa
F1	315	340	350	360
F2	316	341	351	361
F3	317	342	352	362
F4	318	343	353	363
F5	319	344	354	364
F6	320	345	355	365
F7	321	346	356	366
F8	322	347	357	367
F9	323	348	358	368
F10	324	349	359	369
F11	389	391	393	395
F12	390	392	394	396

## 7. Příklady realizace funkčních celků

V této kapitole nalezne čtenář několik příkladů funkčních celků realizovaných v jazyce LEDA. Jde o celky menšího rozsahu, které by měly ilustrovat především použití některých standardních funkcí jazyka a jeho operátorů. Jednotlivé příklady zahrnují pouze fragmenty zdrojového textu, bez uvedení všech potřebných deklarací proměnných a případně ekvivalencí řetězců. Příklady jsou doplněny blokovými schématy ve formě, kterou považuje autor za srozumitelnou pro většinu čtenářů obeznámených alespoň v hrubých rysech s problematikou zpracování signálů v systémech pro přímé řízení technologických procesů.

### Astabilní klopný obvod

Klopný obvod lze sestavit ze dvou sériově řazených časových členů se "zápornou" zpětnou vazbou (negací ve zpět.vazbě):

```
!Vystup, T1 > SDR, T2 > DSR > Vystup;
```

Přitom T1+1 je počet period doby trvání impulsu (stav "1" proměnné Vystup) a parametr T2+1 určuje počet period doby trvání mezery mezi impulsy.

Jestliže zavedeme zpětnou vazbu jen přes jeden zpoždovací člen, získáme astabilní klopný obvod generující impulsy délky jedné periody zpracování té sítě, ve které je tento člen zařazen:

```
!Vystup, T2 > DSR > Vystup;
```

Jiná varianta generátoru pulsů umožňuje naopak seřizování délky výstupních impulsů při mezeře délky jedné periody zpracování dat:

```
!Vystup, T1 > SDR > Vystup;
```

V některých případech se uplatní i nejjednodušší kmitavý obvod:

```
!Vystup > Vystup;
```

### Startovaný klopný obvod

Například pro generaci výstupu pro akustickou signalizaci dosud nekvitované poruchy lze použít tuto variantu startovaného klopného obvodu:

```
NekvitPor & !Pomlka > Houkej;  
Houkej, T1 > SDR > Pomlka;
```

### Třípolohový regulátor polohy pohonu

Rozdíl žádané a skutečné polohy je zpracován třípásmovým komparátorem CMP se dvěma BOOL výstupy tvořícími povely VÍCE (Up) a MÉNĚ (Down).

```
PW - PX, -dL, dH, HystP, > CMP > Down, Up;
```

Hodnoty parametrů dL a dH určují pásmo necitlivosti servosmyčky, HystP určuje hysterezi společnou pro spínání obou směrů pohybu pohonu.

### Třípolohový regulátor teploty

Skutečná teplota TX je porovnávána s dolní a horní mezní teplotou Tmin a Tmax. Pro TX <= Tmin dává funkce CMP výstupní signál PVTVUp s významem "ventil topné vody otvírat"; pro TX >= Tmax dává funkce CMP PVTVDn=1 s významem "ventil topné vody zavírat". Pro teplotu TX v rozmezí Tmin až Tmax jsou oba výstupy CMP nulové. Hystereze, jako další vstupní parametr CMP, je v tomto případě nezadaná, což má stejný účinek jako její nulová hodnota.

```
TX, Tmin, Tmax,, > CMP > PVTVUp, PVTVDn;
```

Pro pevně zadané hodnoty Tmin a Tmax můžeme psát přímo např.:

```
TX, 52°C $TT, 56°C $TT,, > CMP > PVTVUp, PVTVDn;
```

přičemž \$TT identifikuje reprezentaci pro teploty (deklarovanou v programu výše).

### Pásmo necitlivosti a saturace

Vstupní signál Vstup má být zpracován tak, aby výstup Vystup měl nenulovou hodnotu jen pro Vstup<<NMin nebo Vstup>>PMin a aby byl omezen tak, že bude platit Vystup>=LoLim a zároveň Vystup<=HiLim.

Požadovaného efektu lze dosáhnout sériovým řazením standardních funkcí DBN (pásmo necitlivosti) a SAT (saturace):

```
Vstup, NMin, PMin > DBN, LoLim, HiLim > SAT > Vystup;
```

Jinou možností je použití standardní funkce NLT:

```
Vstup,  
LoLim+NMin, LoLim,  
NMin, 0,  
PMin, 0,  
HiLim+Pmin, HiLim > NLT > Vyst;
```

což je méně přehledné, ale obecnější.

### Normalizace, filtrace a komparace na proměnné mezní hodnoty

Vstupní signál TX je nejprve zpracován standardní funkcí IPR, která pro zadaný typ vstupu (typ=2), tj.dvoubytový unipolární signál, provede normalizaci do formátu FIXP proměnné. Signál je dále filtrován dolní propustí 1. řádu s časovou konstantou 120s a komparován s momentální hodnotou dolní meze určené proměnnou LoLimX a horní meze určené proměnnou HiLimX. Výsledná dvouhodnotová proměnná OutLim bude mít hodnotu 1 v případě, že filtrovaný TX je mimo rozsah (LoLimX, HiLimX).

```
TX, 2 > IPR, 120s $ > FIL, LoLimX, HiLimX, HystX > CMP > OutLim;
```

Pokud není cílem zpracování TX pouze komparace na meze, ale filtrovaný signál TXf budeme dále v programu používat, je vhodné pozměnit uspořádání takto:

```
TX, 2 > IPR, 120s $ > FIL > TXf;  
TXf, LoLimX, HiLimX, HystX > CMP > OutLim;
```

### Kontrola rozběhu čerpadla

Nejpozději za 5s po vydání povelu PumpON k zapnutí motoru čerpadla musí být dosaženo minimální tlakové diference návěštěné signálem PumpRun. Není-li po uplynutí 5s a kdykoli později po dobu trvání PumpON signál PumpRun ve stavu 1, dojde k nahození klopného obvodu svýstupemPumpFail. Tento klopný obvod lze shodit signálem ResPump (kvitace poruchového stavu).

```
PumpON, 5s $ > DSR & !PumpRun > SET > PumpFail;  
ResPump > RES > PumpFail;
```

Může-li být motor čerpadla zapnut i pro PumpON=0 (v režimu lokálního řízení při PumpLoc=1) a porucha PumpFail je indikována i při přehřátí motoru čerpadla (při TPump=1) je třeba schéma upravit např. takto:

```
PumpON, 5s $ > DSR & !PumpRun & !PumpLoc | TPump > SET > PumpFail;  
ResPump > RES > PumpFail;
```

Signál PumpRun z čidla tlakové diference podle předcházejícího příkladu je nevěrohodný v situaci, kdy není zapnuto čerpadlo (PumpON=0) nebo uplynulo více než 5s od vypnutí čerpadla, ale přesto je vstupní signál PumpRun=1. Je proto vhodné doplnit program např.takto:

```
PumpON, 5s $ > !SDR & PumpRun > SET > SdpFail;
ResSdp > RES > SdpFail;
```

### Sepnutí zpožděné od náběžné hrany vstupního impulsu

Jen na první pohled se může zdát, že se jedná o případ, kdy lze jednoduše použít funkci zpožděného náběhu signálu DSR. Je-li totiž vstupní signál impulsní a jeho doba trvání není zaručeně větší, než zadané zpoždění výstupu od jeho náběžné hrany, byl by výstup předčasně shozen.

Problém řeší např. následující kombinace funkcí:

```
Vstup, TZN > MSF > NIMP > Vystup;
```

pro případ, kdy má být výstupem impuls jednotkové šířky, nebo:

```
Vstup, TZN > MSF > NIMP > SET > Vystup;
```

pro případ, kdy má být výstup bistabilní (shození musí být zajištěno jinde). Parametr TZN určuje zpoždění náběžné hrany výstupu.

### Sepnutí zpožděné od sestupné hrany vstupního impulsu

Mechanicky by bylo možné od předcházejícího příkladu odvodit pro požadovaný impulsní výstup:

```
Vstup > NIMP, Tzs > MSF > NIMP > Vystup;
```

výhodnější ale je použití funkce zpožděného odpadnutí SDR:

```
Vstup, Tzs > SDR > NIMP > Vystup;
```

### Sekvenční řízení uzavíracích ventilů

Předpokládejme, že výskyt startovacího impulsu Start má vyvolat následující sekvenci akcí:

1. Vyslat povel k uzavření ventilu V1,
2. Po dosažení koncové polohy V1Uzavren a prodlevě T1 vyslat povel k otevření ventilu V2.
3. Po dosažení koncové polohy V2Otevren a prodlevě T2 vyslat povel k opětovnému uzavření vent. V2.
4. Po dosažení koncové polohy V2Uzavren a prodlevě T3 vyslat povel k otevření ventilu V1.

Jinou formou zadání může být časový diagram.

Je-li každý uzavírací ventil řízen jediným výstupním BOOL signálem, lze algoritmus popsat v jazyce LEDA např. takto:

```
Start > RES > V1Otevri; /* V1 Zavirej! */
V1Uzavren, T1 > DSR > IMP > SET > V2Otevri; /* V2 Otevirej! */
V2Otevren, T2 > DSR > RES > V2Otevri; /* V2 Zavirej! */
V2Uzavren, T3 > DSR > IMP > SET > V1Otevri; /* V1 Otevirej! */
```

Lze si povšimnout, že povel k zavírání ventilu V2 je odvozen od úrovně výstupu zpoždovacího členu DSR se vstupem V2Otevren, zatímco povely k otvírání V2 a V1 jsou odvozeny pomocí funkcí IMP pouze od náběžných hran výstupů příslušných zpoždovacích členů. Důvod je zřejmý - např. v době, kdy je V1 uzavřen se má V2 otevřít a opět zavřít, je tedy nahození výstupu V2Otevri nutné odvodit od náběžné hrany výstupu DSR.

Poněkud jiná situace nastává pro případy, kdy jsou uzavírací ventily řízeny dvojicí signálů Otevirej-Zavirej. Předpokládejme, že vstupem do automatu je mj. signál RezimZ1 jako požadavek uzavření ventilu V1 a následného

otevření ventilu V2; přejde-li vstup RezimZ1 naopak do stavu 0, má být nejprve ventil V2 uzavírán a po jeho uzavření má být otevřen ventil V1. Dále předpokládáme, že při uzavírání ventilů má příslušný signál Zavirej ještě např. 20s po přijetí signálu Uzavren od koncového spínače.

Zápis v jazyce LEDA může vypadat např. takto:

```
!V1Uzavren          & RezimZ1 > SET > V1Zavirej; /*zacat.zav. */
  V1Uzavren,20s $ > DSR | !RezimZ1 > RES > V1Zavirej; /*konec zav. */
!V1Zavirej          & RezimZ1 > SET > V2Otevirej; /*zacat.otev.*/
  V1Otevren         & !RezimZ1 > RES > V2Otevirej; /*konec otev.*/
!V2Uzavren          & !RezimZ1 > SET > V2Zavirej; /*zacat.zav. */
  V2Uzavren,20s $ > DSR | RezimZ1 > RES > V2Zavirej; /*konec.zav. */
!V2Zavirej          & !RezimZ1 > SET > V1Otevirej; /*zacat.otev.*/
  V2Otevren         & RezimZ1 > RES > V1Otevirej; /*konec otev.*/
```

Sekvenční automat se vlastně skládá ze čtyř klopných obvodů pro jednotlivé povely Zavirej-Otevirej. Změna signálu RezimZ1 do stavu 1 má okamžitý účinek na ukončení uzavírání ventilu V2, přechod do 0 na ukončení uzavírání ventilu V1. Výstupní povely jsou shazovány do stavu 0 po dosažení koncové polohy, resp. za 20s od hlášení V.Uzavren.

### Obecný sekvenční automat

Je-li sekvenční automat zadán formou stavového diagramu, nebo lze na tuto formu od zadání přirozeně přejít, může být výhodné realizovat takový automat v jazyce LEDA pomocí příkazu SWI. Jestliže má mít automat nejvýše 256 stavů, můžeme deklarovat proměnnou STAT BYTE StavAutomatu a jednotlivé stavy automatu očíslováme (000 .. 255). "Kostru" automatu tvoří příkaz SWI s řídicí proměnnou StavAutomatu:

SWI(StavAutomatu)

```
{ 000: Příkaz000;
  001: Příkaz001;
  .....
  xyz: Příkazxyz;
};
```

Příkazy (složené) Příkaz000, Příkaz001, ... popisují pro jednotlivé stavy automatu generaci výstupů a vyhodnocení podmínek přechodu do jiných stavů, které obecně končí nastavením nové hodnoty proměnné StavAutomatu. Při následujícím vyvolání tohoto příkazu SWI bude při StavAutomatu==xyz vykonán Příkazxyz.

Každý složený příkaz Příkaz... může zahrnovat přiřazovací příkazy s výrazy na levých stranách (kombinační logické funkce s argumenty tvořenými jednak vstupy automatu, jednak výstupy časových členů typu DSR nebo MSF) a s výstupními signály automatu na pravých stranách a dále případné (obecně podmíněné) starty popř. rušení různých prodlev a podmíněné nastavení hodnoty proměnné StavAutomatu. Podmínky přechodu do různých následujících stavů automatu lze vyhodnocovat příkazy IF ... ELSE nebo dalšími (vnořenými) příkazy SWI(Vstupy), kde proměnná vstupy může podle typu BYTE resp. FIXP koncentrovat max. 8 resp. max 16 dvouhodnotových vstupů automatu. Hodnotu proměnné Vstupy je potřebné naplnit před vyhodnocením ve vnořeném SWI nebo společně pro všechny stavy automatu před vnějším SWI (výhodné je užití standardní funkce CPS). Konstanty komparované v exekutivě SWI postupně s momentální hodnotou proměnné StavAutomatu lze přehledně zapisovat s využitím standardních reprezentací \$B popř. \$BB (tj. ve tvaru např.: 00101101 \$B). Skupině dvouhodnotových výstupů lze rovněž přiřazovat konstanty zadané pomocí \$B nebo \$BB a zápis vnořeného příkazu SWI pak může nabýt formy blízké pravdivostní tabulce:

SWI(Vstupy)

```
{ 00000000 $B: { 01001000 $B > Vystupy; ... };
  00000001 $B: { 00001000 $B > Vystupy; ... };
  00000010 $B: { 00000110 $B > Vystupy; ... };
```

```
};
```

Pro vysílání výstupů lze využívat také standardní funkci [DCPS](#).

Použití vnořeného příkazu SWI naznačeným způsobem je výhodné v případech, kdy pro daný stav automatu existuje vždy jen poměrně malý počet povolených kombinací vstupů, které určují požadavky na změny výstupního vektoru a ostatní kombinace nastat nemohou, popř. je lze ošetřit jednotným způsobem.

Výhodné může být ve SWI(Vstupy) nejprve rozlišit případy, kdy výstup automatu ovlivňují určité kombinace hodnot všech vstupních veličin a v [DEFAULT](#) větvi tohoto SWI dořešit ostatní případy, které lze výhodněji podchytit BOOLskými výrazy.

### Klasifikace úrovně signálu - absolutní četnost výskytu daných hodnot

Předpokládejme, že je požadováno vyhodnotit v nějakém časovém intervalu absolutní četnost výskytu „padnutí“ vstupního signálu do některého z „třídních intervalů“. Pokud mají tyto intervaly různé velikosti, nabízí se použití řady sériových kombinací [CMP-CNT](#) (komparátor-čítač). Jsou-li však všechny intervaly stejné šíře, lze volit elegantnější řešení s absolutními četnostmi uloženými např. v jednorozměrném poli `Statist[0..22]` (pro 23 třídních intervalů):

```
Vstup/SirkaIntervalu > Poradi;  
IF(Poradi<=21) Statist[Poradi]+1 > Statist[Poradi]  
ELSE Statist[22] +1 > Statist[22];
```

### Saturace výstupu astatických regulátorů – potlačení „přeintegrování“

Regulátor [PID](#) tvořený stejnojmennou standardní funkcí jazyka LEDA zajišťuje saturaci výstupu na okamžitých hodnotách parametrů `LoLimY`, `HiLimY`, které mohou být konstantní nebo algoritmicke řízené. Konstrukce funkce `PID` vylučuje nežádoucí nárůst integrační složky za mezí saturace, ale nemůže bez přidavných opatření respektovat skutečný (momentální) rozsah linearity navazujícího systému (navazujícího regulačního obvodu kaskády nebo servosmyčky).

Pokud jsme v situaci, kdy "dorazy" řízeného systému nejsou dostatečně přesně známy nebo se předpokládají jejich víceméně náhodné změny, je vhodné uvažovat o možnosti konstrukce "klasifikátoru", který rozpoznává situaci saturace a o zpracování jeho `BOOL` výstupů "SatLo", "SatHi". Lze např. doporučit použít na místě parametrů `LoLimY`, `HiLimY` funkce [SEL](#), jejichž jeden vstup je konstantou zvolenou vně nejširšího možného pásma linearity a druhý vstup je totožný s výstupem regulátoru. Dokud není rozpoznán stav saturace navazujícího systému, jsou `SatLo` a `SatHi` nulové a `LoLimY`, `HiLimY` vymezují dostatečně široký interval povolených výstupů regulátoru. Je-li např. `SatHi==1`, dojde ke zmrazení výstupu na momentální úrovni až do odpadnutí signálu `SatHi` nebo poklesu velikosti výstupu.

Klasifikátor stavu saturace odpadá v případě dostupnosti signálů od koncových spínačů; velmi jednoduše lze někdy saturaci vyhodnocovat sledováním diferencí řízené veličiny, např. takto:

```
(Signal>DIF>ABS)<= Delt, TL > DSR > Klid;  
Klid & Vice > SatHi;  
Klid & Mene > SatLo;
```

kde

`Signal` je řízená veličina, `Delt` je tolerance pro potlačení vlivu šumu, `TL` je časový limit na "rozjezd či uklidnění", `Vice` a `Mene` jsou povely, které mají v pásmu linearity ovlivňovat veličinu `Signal`.

## 8. Souhrn upozornění, rad a doporučení

Tato kapitola shrnuje různé informace, které většinou nelze považovat za nezbytnou součást popisu jazyka v jeho obecnosti, ale přesto mohou být uživateli užitečné zvláště v období, kdy začíná psát své první programy v jazyce LEDA.

### 8.1 Řazení sítí a viditelnost proměnných

Lze doporučit následující pořadí sítí v programu:

1. síť zahrnující deklarace uživatelských funkcí
2. první síť podřízené řídicí stanice s periodou shodnou se základní periodou PERIOD
3. další cyklické sítě podřízené stanice s periodami většími nebo shodnými s PERIOD
4. MASTER síť

Sítě podle 3. a 4. mohou být pochopitelně řazeny libovolným způsobem.

Uvedené pořadí sítí je vhodné z hlediska viditelnosti proměnných - MASTER sítě uvedené v programu níže mohou zpracovávat vedle všech implicitních proměnných ležících vždy pouze v paměti podřízené stanice i všechny globální statické proměnné deklarované v sítích 2. a 3., tedy proměnné uvedené v deklaracích předcházejících levé složené závorky prvních bloků těchto sítí.

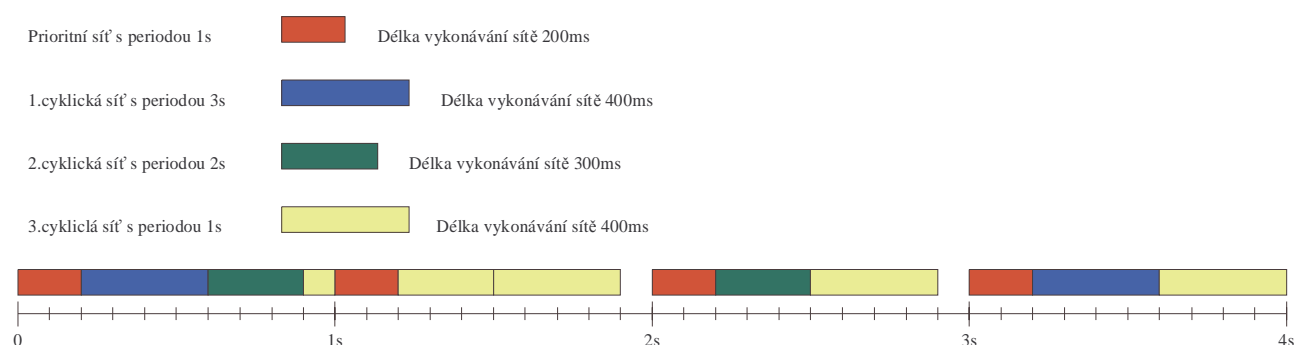
MASTER sítě mohou proměnné alokované překladačem do paměti podřízené stanice zpracovávat a přepisovat, obráceně však není přípustné - bez užití rozšířených adres a linkování "projektu distribuovaného systému" - odkazovat se ze sítí 2. a 3. na globální proměnné sítě MASTER.

Rovněž při deklaraci proměnných alokovaných v paměti PC je třeba mít na mysli základní pravidlo viditelnosti proměnných:

Proměnné je nutné nejprve deklarovat (výše v programu) a teprve poté užívat.

Program pro PC nemá žádné vlastní implicitní proměnné! Všechny proměnné určené pro činnost MASTER sítí je nutno deklarovat, a to uvnitř těchto sítí, tedy za jejich BEGIN; globální proměnné viditelné ze všech následujících MASTER sítí musí být deklarovány před levou složenou závorkou prvního bloku MASTER sítě.

Na následujícím obrázku je znázorněno spuštění několika sítí různých délek a různých period



Z obrázku je zřejmé, že v prvním cyklu byly spuštěny všechny sítě, ale 3.cyklická síť byla přerušena prioritní sítí. V druhém cyklu byla dokončeno první spuštění 3.cyklické sítě a potom byla znovu spuštěna stejná síť (spouští se každou periodu).

Pokud dojde k přetížení exekutivy v prioritní síti (prioritní síť nebyla ukončena do počátku další periody), má to za následek reset stanice, přičemž tato událost se promítne do obsahu chybových čítačů stanice (viz. Technické příručky jednotlivých stanic).

Při přetížení cyklických sítí nedojde k resetu stanice a sítě jsou spuštěny tak rychle, jak to jejich vykonávání umožňuje. Přetížení cyklických sítí je opět indikováno v chybových čítačích stanice.

Délky vykonávání jednotlivých sítí se určují pomocí systémových proměnných TIMEX a MILS. Obě tyto proměnné jsou cyklicky inkrementovány systémem, rozdíl je, že zatímco TIMEX je inkrementován trvale, MILS se inkrementuje pouze v době, kdy není zpracovávána prioritní síť. To umožňuje měření doby zpracování i u cyklické sítě, která je přerušena prioritní sítí.

Měření doby zpracování sítě probíhá tak, že na počátku měřeného úseku programu se nastaví do příslušné proměnné (TIMEX pro prioritní síť nebo celkovou dobu vykonávání programu nebo MILS pro cyklickou síť) nulová hodnota a na konci měřeného úseku se obsah proměnné překopíruje do pomocné proměnné. Hodnota pomocné proměnné potom udává délku měřeného úseku v milisekundách.

## 8.2 Volba period zpracování dat

Základní perioda zpracování dat zadávaná (v [s] jako desetinné číslo) v záhlaví programu za klíčovým strojem PERIOD je základní periodou zpracování dat v podřízené řídicí stanici, nikoli v PC!

Hodnotu PERIOD volíme mj. s uvážením časové rezervy na zpracování prioritních zpráv v řídicí stanici - prověření výkonu je potřeba provádět pro co největší objem dat požadovaných z PC pro zpracování v MASTER sítích, aktualizaci zobrazovaných hodnot a při přenášení maximálního počtu povelů modifikujících hodnoty proměnných ležících v paměti podřízené řídicí stanice.

Skutečnou délku interpretace jednotlivých sítí aplikačního programu nebo i menších úseků programu lze vyhodnocovat pomocí systémových proměnných TIMEX a MILS.

Základní perioda zpracování dat v MASTER sítích se v programu nezadáva - je při překladu určena jako největší společný dělitel period všech MASTER sítí zadávaných v záhlaví těchto sítí v celých sekundách. S takto určenou periodou - nebo v jejím násobku - probíhá komunikace s podřízenou stanicí. Volba delších period MASTER sítí obecně zvyšuje časovou rezervu pro komunikaci a interpretaci aplikačního programu v PC a zpomaluje aktualizaci dynamické části technologických schémat. Rychlost reakce na zásahy operátora (vyřizované prioritně) kolísá v závislosti na délce zprávy přenášené v okamžiku zásahu operátora. Perioda MASTER sítí by měla mj. vyhovět poměrně časově náročným operacím v zápisu do diskových souborů (OUTF).

Jsou-li v master-sítích využívány (exekučně) proměnné alokované v podřízené stanici (implicitní nebo deklarované v síti bez klíčového slova MASTER), je start exekutiv v PC podmíněn ukončením příslušného sběru dat! Počet těchto proměnných omezuje použitelnou minimální periodu interpretace exekutiv v operátorské stanici.

Periodický přenos několika BOOL proměnných je daleko náročnější, než např. přenos jediné proměnné typu BYTE nebo FIXP - z hlediska přenosových kapacit je proto výhodné sdružovat BOOL proměnné a z jiné stanice se odkazovat na proměnnou BYTE či FIXP přiřazenou (exekučně) do proměnné stejného typu (schránky plněné komunikačním subsystémem), se kterou mohou být ekvivalencí adres (EQU) svázány opět BOOL proměnné.

Je vhodné si uvědomit, že parametry reprezentací tvořené proměnnými alokovanými v podřízené řídicí stanici vyvolávají požadavky na sběr dat podmíněný volbou příslušného výřezu zdrojového textu, a to pro jednu proměnnou vyskytující se v několika reprezentacích s markerem v daném výřezu opakovaně! To znamená, že např. N textů (\$GTEXT) s markery v jednom technologickém schématu s barvou určenou jediným indexem režimu vyhodnocovaným v podřízené stanici vyvolává pro periodickou aktualizaci N dotazů na jedinou proměnnou typu BYTE. Tato zdánlivá nadbytečnost požadavků na komunikaci je vyvolána požadavkem na zobrazitelnost libovolných výřezů zdrojového textu - textem lze "narolovat" podle přání operátora na zvolený řádek a soubor aktualizovaných reprezentací je proto velmi variabilní. Úsporu komunikačního času lze docílit tím, že místo proměnné podřízené stanice použijeme jako parametr reprezentací proměnnou deklarovanou v některé master-síti a zařadíme do této sítě jediný přiřazovací příkaz (exekuční) s původní zdrojovou proměnnou (ze SLAVE) na levé straně přiřazení. Místo N požadavků na přenosy vyvolané bezprostředně vizualizací vzniká tímto opatřením jediný požadavek na periodický přenos vyvolaný požadavky exekutivy.

### Pozor!

Pouze síť pracující v základní periodě PERIOD resp. v základní periodě zpracování dat v PC mohou pracovat se systémovými proměnnými START a RESTART v podřízené stanici resp. pouze START v PC a generovat odvozené signály pro řízení inicializace v sítích spouštěných s delšími periodami.

## 8.3 Rychlost interpretace

Programátor může v některých případech zásadním způsobem ovlivnit časovou náročnost interpretace - určitý problém lze řešit s užitím různých programátorských technik, které kladou rozdílné nároky na čas procesoru a kapacitu operační paměti.

Obecně lze konstatovat, že:

Používání indexovaných proměnných s indexy tvořenými (nekonstantními) výrazy zvyšuje časové nároky, naproti tomu např. rozepisování cyklů vede k rychlejší interpretaci, ale většímu rozsahu přeloženého programu (uživatelské databáze). Časové nároky poněkud zvyšuje také kontrola rozsahu indexů (Range Checking) prováděná za běhu programu, přesto ji lze pro zvýšení odolnosti vůči chybám aplikačního programu doporučit (požadavek se zadává v Options vývoj.prostředí COLEDA).

Používání deklarovaných funkcí zvyšuje dobu exekutivy, a to tím více, čím větší počet hodnot skutečných vstupních a výstupních parametrů musí být přenášeno do a ze zásobníku. Opakování částí programu může být z tohoto hlediska výhodnější, i když méně přehledné a paměťově náročnější.

U složitějších logických automatů nebo při vyhodnocování většího počtu aritmetických výrazů může být užitečné nalézt vícenásobně použitelné mezivýsledky - opakování některých výrazů sice může zlepšit čitelnost programu (v daném místě programu je vidět vše potřebné), ale platí se časem interpretace.

Práce s BOOL proměnnými je poměrně zdoluhavá (maskování jednotlivých bitů) - je-li např. určitý režim určen skupinou nejvýše osmi resp. šestnácti dvouhodnotových proměnných, je daleko rychlejší testovat určitou hodnotu proměnné BYTE resp. FIXP deklarované ve stejné paměťové oblasti jako BOOL proměnné (pomocí EQU), než vyhodnocovat BOOLský výraz (jako logický součin). Rovněž nastavit číslo určující režim je mnohem rychlejší, než nahodit resp. shodit všechny zúčastněné BOOL proměnné. Speciálně např. test nulovosti slova může nahradit až 15 logických součtů.

Některé standardní funkce mohou - již svým názvem - svádět k neadekvátnímu užívání. Např.:

místo:

```
Ab > SET > Bc;  
!Ab > RES > Bc;
```

stačí pouze:

```
Ab > Bc;
```

místo:

```
Podminka, 1, 0 > SEL > Vystup;
```

stačí pouze:

```
Podminka > Vystup;
```

## 8.4 Inicializace proměnných

Při studeném startu stanice jsou před spuštěním vlastní exekutivy samočinně některé statické proměnné nastaveny na hodnoty, které jim byly přiřazeny komunikačním přiřazením při překladu programu:

```
IdentifikátorProměnné < Konstanta $ Repräsentace;
```

nebo

```
IdentifikátorProměnné < Repräsentace $ Konstanta;
```

Identifikátor Repräsentace a popř. i symbol \$ lze vypustit (zadání hodnoty v % popř. v "bitech").

V podřízené stanici mohou být takto inicializovány pouze proměnné tvořící prvky polí implicitních proměnných s identifikátory:

```
S, P, W, F
```

Naproti tomu ve stanici master mohou být obdobným způsobem inicializovány všechny statické proměnné deklarované v MASTER sítích. Z toho plyne, že inicializované proměnné alokované v paměti PC mohou být jakéhokoli povoleného typu (avšak vždy STAT), zatímco u proměnných podřízené stanice jsou k dispozici bezprostředně jen typy:

FIXP (S, P, W), BOOL (S, P, W) a FLTP (F).

Inicializované proměnné typu CHAR, řetězce CHAR #n a pole různých typů lze v podřízené stanici zařadit formou ekvivalentní adresace vůči implicitním proměnným. Toto překrytí paměti se zadává v rámci deklarace podle následujících vzorů.

proměnná:

```
STAT CHAR Promenna EQU Pmn;
```

řetězec:

```
STAT CHAR #d Retez EQU Wmn;
```

pole:

```
STAT CHAR ARRAY PoleCh[i..j] EQU Smn;
```

```
STAT BYTE ARRAY PoleBy[i..j] EQU Pmn;
```

přičemž d, m, n, i, j jsou číselné konstanty určující délku řetězce, index prvku pole implicitních proměnných a rozsah indexů deklarovaného pole.

Inicializaci hodnot proměnných (jednoduchých i polí) lze v programu zadávat také v rámci deklarací - na rozdíl od skalárních a vektorových parametrů nelze za běhu v deklaracích modifikovat hodnoty proměnných a měnit směr přiřazení (zadání/zobrazování).

Počáteční hodnoty je ovšem možné libovolným proměnným přiřazovat také exekucním přiřazením ve větvi aplikačního programu podmíněně nastavením systémových proměnných RESTART nebo START:

```
IF (START|RESTART) { Konst 1 > Promenna 1;
                    Konst 2 > Promenna 2;
                    ...
                    };
```

## 8.5 Výměna dat mezi stanicí MASTER a SLAVE

Připomeňme nejprve, že v sítích MASTER lze užívat (ve výrazech) proměnné deklarované ve stanici SLAVE (stanice spojeny linkou nebo sběrnici) a těmito proměnným lze exekučně přiřazovat výsledky zpracování dat. Rovněž v sítích podřízené stanice lze exekučně užívat proměnné deklarované v MASTER sítích a přiřazovat jim výsledky vyhodnocení výrazů.

Objasněme podrobněji mechanismus výměny dat mezi stanicemi - jeho znalost může zabránit různým nekorektním obrátům při tvorbě aplikačních programů.

Každá proměnná odkazovaná pro exekuční užití z cizí stanice má překladačem přiřazeny dvě paměťové buňky (skupiny buněk s rozsahem určeným typem proměnné):

- "originální" buňky v paměti stanice, v níž byla proměnná deklarována,
- "schránku" pro hodnoty dané proměnné tvořenou buňkami alternativní stanice.

Komunikační subsystém modulu TOLEDA vždy před spuštěním interpretru XEDA1 zajistí:

- Vyslání hodnot schránek pro proměnné stanice SLAVE, které se vyskytly na pravé straně přiřazovacích příkazů (v "cíli") v sítích MASTER.
- Vyslání hodnot proměnných MASTER sítí, které se vyskytují na levé straně exekučních přiřazení realizovaných ve stanici SLAVE, do příslušných schránek ve stanici SLAVE.
- Vyžádání hodnot proměnných ze schránek ("cílu") exekučních přiřazení stanice SLAVE do "originálních" buněk v MASTER.

- Vyžádání hodnot proměnných stanice SLAVE vyskytujících se na levé straně přiřazení (ve "zdroji") realizovaných v MASTER sítích a zápis přijatých hodnot do příslušných schránek v MASTER.

Jeden z interpretů operuje na originálních buňkách, druhý na kopiích v příslušných schránkách. Interpretace nejsou synchronizovány!.

Z uvedeného vyplývá, že s proměnnými cizí stanice nelze zacházet stejně jako s vlastními proměnnými.

Uveďme příklad signálové vazby stanic MASTER a SLAVE, ve kterém se projeví naznačené problémy.

Předpokládejme, že ve SLAVE stanici je deklarována proměnná M s významem "StartAkce" a ve stanici MASTER běží algoritmus, který určí okamžik, kdy je potřeba tuto blíže neurčenou akci ve SLAVE odstartovat. Pokud bychom pominuli uvedená fakta o komunikacích mezi stanicemi, zdálo by se jako postačující zvolit toto

chybné řešení:

```
MASTER          IF(Cond) 1 > M;
SLAVE           IF(M) { 0 > M; ... /* Akce */ }
```

Důsledky jsou zřejmé: je-li jednou splněna podmínka IF(Cond) v MASTER síti, je schránka pro M ve stanici MASTER nastavena "navždy" do "1" a je trvale cyklicky přenášena do originální buňky M. Akce se provádí opakovaně s periodou určenou periodou aktualizace dat, tedy přibližně s periodou vykonávání MASTER sítě (je-li perioda zpracování podmínky IF(M) menší nebo rovna periodě MASTER sítě).

Jedním z možných korektních řešení je:

```
MASTER          Cond > IMP > M;
SLAVE           IF(M>IMP) { ... /* Akce */ }
```

místo stavu proměnné M se pracuje s náběžnými hranami.

Pokud z nějakých důvodů chceme zachovat podmíněný příkaz IF v MASTER, můžeme psát také např.:

```
MASTER          IF(Cond) { !M > M; ... }
SLAVE           IF(M>DIF) { ... /* Akce */ }
```

kde každá hrana signálu M startuje Akci.

## 8.6 Ladění aplikačních programů

### Odstraňování syntaktických chyb

Některé syntaktické chyby nejsou překladačem lokalizovatelné tak, aby bylo možno snadno je nalézt a odstranit - např. chybějící uzavírací příkazová závorka způsobí úplné "zmatení" úrovní vnoření, které překladač zjistí až na konci sítě nebo celého programu. Jestliže byl od posledního úspěšného překladu program modifikován na větším počtu míst, může být oproti prohlížení textu vhodnější opakovat pokusně překlad s vyřazením některých sítí, bloků, složených příkazů nebo i menších úseků programu dvojicemi komentářových závorek. Jestliže syntaktická analýza proběhne bez chyby, byla chyba uvnitř vyřazené části programu a obráceně. Při hledání syntaktické chyby můžeme komentářovými závorkami bez obav vyřadit i některé deklarace proměnných či funkcí - sémantické chyby nás v této fázi nezajímají. Pokusné dvojice komentářových závorek je vhodné výrazně označit těžko zaměnitelným způsobem, např.: /\*!!! !!!\*/

### Odlad'ování programů po částech

Při odlad'ování větších programů je vhodné postupovat po menších celcích (např. sítích), momentálně nezajímavé části programu uzavírat do komentářových závorek - překlady mohou probíhat mnohem rychleji než pro kompletní program a na omezené ploše hledáme chyby snadněji.

### Užívání skalárních parametrů

Jazyk LEDA poskytuje uživateli možnost zahrnout do svého programu tzv. skalární parametry - prostředky pro zobrazování zpracovávaných vstupních proměnných, výstupů a mezivýsledků a pro modifikace vstupů. Program si tedy může "nést ve svém zdrojovém textu" informace nahrazující do jisté míry parametry zadávané při ladění programů v běžných univerzálních jazycích ladicím programům. Hojně užívání skalárních parametrů a standardních reprezentací lze doporučit, je však vhodné upozornit na některá omezení jejich použitelnosti.

Jsou-li v částech programu, které v daném okamžiku nejsou zpracovávány (není splněna podmínka IF...ELSE nebo ve SWI) umístěny skalární parametry, probíhá zobrazování skutečných hodnot proměnných stejně jako u reprezentací zapsaných v momentálně aktivních větvích programu.

### **Sledování hodnot modifikovaných s krátkou periodou**

Změny hodnot proměnných přepisovaných v rychlejších sítích řídicí stanice pracující v reálném čase lze korektně zjišťovat jen pomocí jejich fixace exekucním přiřazením do pomocných proměnných resp. polí, popř. lze do programu dočasně zařazovat pomocné komparace s výstupem zpracovávaným funkcí SET nebo čítačem CNT. Výhodné může být rovněž použití funkce SRG (posuvný registr), která může v pomocném jednorozměrném poli posouvat při každém vyvolání starší vzorky sledované veličiny a celý soubor těchto vzorků lze pak pozorovat bez ohledu na periodu exekutivy. Funkce SRG může být vyvolána na několika místech programu i se stejným výstupním polem a různými vstupními veličinami. V jednotlivých periodách se pak vytvářejí "záznamy", které lze v odpovídajícím formátu (jako tabulku) také zobrazovat.

### **Autonomní ladění na PC**

V počátečním stádiu ladění je obvykle výhodné pracovat jen autonomně na PC, to znamená, že všechny sítě programu musí být opatřeny klíčovým slovem MASTER. Periodu zpracování je třeba zpočátku volit spíše větší, aby exekutiva vyplnila jen část periody a ve zbytkovém čase mohly být vykonány všechny potřebné operace spojené s vizualizací kteréhokoli výřezu zdrojového textu - aktualizací zobrazovacích reprezentací a se zpracováním vstupů operátora z klávesnice a myši. Větší perioda zpracování umožní samozřejmě také snazší sledování většího počtu zobrazovaných hodnot.

Zpočátku není vhodné přehánět hustotu zápisu algoritmů. Vhodné je používat méně komplikované výrazy s větším počtem zobrazitelných mezivýsledků. Po úspěšném odzkoušení lze v případě nutnosti (zkrácení výsledného souboru SED) zrušit zbytečná zobrazení a zvláštní statické proměnné pro mezivýsledky.

### **Podmíněné a jednorázové spouštění sítí**

Jednotlivé sítě ověřovaného programu mohou být spuštěny až po odblokování příslušné BOOL proměnné "Enable sítě" zadané v jejím záhlaví. Postupně manuální odblokování sítí je významné např. v situaci, kdy se interpretační program po spuštění zhroutí a je proto nutné lokalizovat kritickou část programu. Proměnná Enable sítě může být generována od manuálního zásahu jako impulsní - síť je potom spouštěna např. jen jednorázově a operátor má dostatek času prohlédnout výsledky zpracování, zadat nové vstupy a případně modifikovat hodnoty některých výstupů určených pro jiné sítě nebo zpracování v následující periodě. Start resp. jednorázové spuštění lze provádět pomocí myši přes zadávací okénko s polohou určenou markerem.

Zvolenou část programu lze rovněž v případě potřeby spouštět podmíněně resp. jednorázově, k tomu je však nutné využít přídatný příkaz IF a testovanou část programu uzavřít mezi příkazové závorky.

Staticky lze zvolenou část programu vyřadit uzavřením mezi komentářové závorky - překladač zpracovává vnoření komentářů.

### **Jednorázové spouštění programu**

Jazyk sám neposkytuje žádné speciální prostředky pro jednorázové spouštění celého programu; povely k jednorázovému nebo N-krát opakovanému spuštění programu v podřízené řídicí stanici, provedení inicializace programu a okamžité zastavení interpretace jsou k dispozici ve vývojovém prostředí COLEDA počínaje verzí C31. Zároveň jsou k dispozici volby pro odpojování vstupů z technologie a výstupů do technologie, jejichž využívání usnadňuje zadávání simulovaných vstupních hodnot (včetně údajů systémových hodin reálného času) a brání nežádoucím akčním zásahům neodladěného programu.

### **Volba period zpracování dat a validita zobrazovaných údajů**

Při spouštění programu v řídicí stanici na simulovaných vstupních datech mimo reálný čas je vhodné zajistit volbou period zpracování dat takové poměry, kdy je každá změna zobrazovaných dat zachytitelná. Je třeba si uvědomit, že exekucní proces v podřízené řídicí stanici probíhá asynchronně s exekutivami v PC, a proto také asynchronně se sběrem dat a přenosem povelových zpráv. U proměnných modifikovaných na několika místech programu můžeme zachytit jen výsledek poslední modifikace nebo - většinou s menší četností - kterýkoli mezivýsledek; je-li perioda sběru dat zhruba dvojnásobná proti periodě zpracování dané sítě, pozorujeme jen všechny sudé nebo liché výsledky, ... .

### **Výstupy z řídicí stanice přerušují interpretaci!**

U proměnných modifikovaných na několika místech programu řídicí stanice mimo prioritní síť není zajištěno, že nebudou vyslány hodnoty před poslední modifikací - data z pole OL vystupují v základní periodě programu PERIOD a výstup dat přerušuje interpretaci příkazů a funkcí aplikačního programu. Pokud tato skutečnost brání správné funkci řídicího systému, je nutné program přepracovat takovým způsobem, aby byla postupně modifikována pomocná proměnná a teprve na závěr ji překopírovat do pole OL, nebo tak, aby postupná modifikace vůbec nevznikla (např. místo několika přiřazení s podmíněním pomocí IF vyhodnotit jediný výraz s výstupem přímo do OL, použít vnořené IF .. ELSE nebo jediný příkaz SWI.

### **Indikace průchodu programu kontrolními body**

Čítač s nezadanými vstupními parametry (čítá vzestupně při každém vyvolání) lze využívat pro testování průchodů určitou větví programu.

Další možností je využívání "breakpointů" (V režimu Run: Alt-B ... zadání "breaku", Alt-K ... zrušení "breaku", Alt-A ... zrušení všech "breaků" stanice SLAVE, Alt-W ... zrušení všech "breaků" v MASTER sítích) - podrobný popis v části COLEDA. U řídicích stanic s menší rezervou kapacity kódové paměti (T2008) "breakpointy" nelze využívat! Stojí-li program v kontrolním bodě, je samozřejmě možné prohlížet a modifikovat hodnoty proměnných, a to proměnných MASTER sítí i proměnných alokovaných v paměti řídicí stanice.

### **Náhradní hodnoty proměnných**

Vstupní proměnné lze přepisovat pomocí komunikačního přiřazení v režimu Input Disconnect (volba v menu Run). U ostatních proměnných je (u řídicích stanic jen pro systém TRONIC 2016) možno provádět tzv. suspendování jednotlivých exekučních přiřazení (překlad musí proběhnout s volbou Options-Compiler-Suspend). Vyřazení přiřazovacího příkazu z exekutivy se provádí v režimu Run kliknutím na příslušný symbol ">", dalším kliknutím na stejné místo je obnoven původní stav. Operátory exekučních přiřazení, která mohou být suspendována, jsou barevně odlišeny od operátorů komunikačního přiřazení (světle modrá proti zelené).