SIMPLE 3

Simple3 - popis jazyka

Vydáno jako volně přístupná a neomezeně šířitelná publikace.

HYPEL březen 2001

1 Úvod

Programovací jazyk SIMPLE3 byl vyvinut k tomu, aby jím mohli být vyjádřeny řídících algoritmy v programovatelných automatech, programovatelných terminálech a kompaktních řídících stanicích vyráběných firmou HYPEL.

Program je prováděn od začátku až do příkazu "END", odkud se běh programu vrací zpět na první instrukci. Program tedy vždy probíhá v jediné smyčce a neumožňuje žádné skoky. Tato pevně předepsaná struktura programové smyčky znemožňuje vznik fatálních chyb, především "bloudění" programu a "zatuhnutí" v nekonečné smyčce.

Vlastní zpracování uživatelského programu probíhá v několika fázích :

- ✓ Vytvoření zdrojového textu programu v jazyce SIMPLE3. Lze provést libovolným textovým editorem, např. editorem Norton Commanderu, editorem WordPad atd... nebo je možné tvorbu zdrojového textu provádět v dodávaném integrovaném prostředí WinSim3 či HypEd.

- ✓ Přeložení zdrojového textu překladačem SIMPLE3. Pokud překlad proběhne bez chyb, vygeneruje překladač soubor *.DNL, obsahující kód pro mikropočítač v automatu. Při použití editorů WinSim3 nebo HypEd se překlad spouští přímo z menu integrovaného prosředí.

- ✓ Natažení ("download") hotového programu do automatu. Provádí se přes převodník RS232->RS485 (PCA1, Link1...), připojený na některý sériový port COM1 až COM4. Download obstarává program LOADER3, který zavede do automatu kód ze souboru *.DNL .Zatažení programu do automatu je též možné přímo s prostředí WinSim3 či HypEd.

1.1 Instalace

Je velmi jednoduchá. Obsah instalační diskety se zkopíruje do libovolného adresáře na pevném disku. Například :

copy A:*.* C:\SIMPLE3

Doporučená organizace dat na disku je tato:

Do domovského adresáře (tedy např. C:\SIMPLE3) zkopírovat obsah instalační diskety. Vlastní zdrojové texty, přeložené *.DNL soubory a listingy mohou být buď

také v tomto adresáři nebo v adresáři jiném, např. podle konkrétního projektu. V druhém případě je vhodné si buď vytvořit BAT soubory na spouštění překladače a podpůrných programů anebo jednoduše přidat do příkazu PATH v souboru AUTOEXEC.BAT cestu do adresáře SIMPLE (tedy v našem příkladu je to C:\SIMPLE3).

Pokud používáme Integrované vývojové prostředí WinSim či HypEd stačí pouze spustit Install.exe na dodaném instalačním disku a vše již proběhne automaticky.

2 Struktura programu

Programové vybavení je členěno do dvou vrstev.

- ✓ První tvoří systémový firmware, umístěný výrobcem v pevné paměti, který řeší veškeré vazby na obvodové vybavení automatu a propojení stanice do sítě.

- ✓ Druhou vrstvou je uživatelský aplikační program, tedy program vytvořený přímo uživatelem, přeložený překladačem a zavedený do paměti automatu.

Obě vrstvy si mezi sebou předávají data prostřednictvím sdílených proměnných, umístěných v paměti RAM.

Po zapojení napájení se spouští systémový firmware. Automat nejprve čeká zhruba 1-2 sekundy na případnou komunikaci po lince RS485 z počítače PC. Pokud detekuje žádost o komunikaci, čeká na další příkazy z linky. Když po tuto dobu žádná žádost nepřijde (na PC neběží žádný program pro komunikaci s automatem, nebo je linka úplně odpojena), automat zahájí přípravy na spuštění uživatelský programu.

- ✓ Zjistí, zda je v paměti zatažen uživatelský program.

- ✓ Zkontroluje, zda souhlasí kontrolní součet paměti programu.

- ✓ Zkontroluje, zda je nastaven příznak AUTORUN pro automatické spouštění programu (tento příznak nastavuje program LOADER3.EXE po každém zatažení programu).

Pokud jsou všechny tři body splněny, automat odstartuje uložený program. Ten je tvořen posloupností příkazů, která se může různě větvit, avšak nikdy nevytvoří cyklus.

Dosáhne-li uživatelský program poslední instrukce skončí a odevzdá řízení opět systémovému firmware. Ten provede potřebné ošetření HW, uložení výstupních proměnných na fyzické výstupy a načtení hodnot fyzických vstupů a opět předá řízení aplikačnímu programu. Z hlediska programátora aplikační program vytváří jedinou nekonečnou smyčku, která se cyklicky vykonává.

Při běhu automatu aplikační program pracuje s proměnnými uloženými v RAM mikropočítače a to i tehdy, jedná-li se o vnější proměnné (vstupy nebo výstupy). V tomto případě se vlastně pracuje nad záchytnými registry, kam jsou přepisovány úrovně (logické i analogové) z fyzických vstupů, resp. odkud jsou obsahy přepisovány na fyzické výstupy. Tento přepis probíhá vždy na začátku programové smyčky. To v praxi znamená, že se během jednoho běhu programové smyčky nemohou změnit obsahy vstupních proměnných a zároveň se jakékoliv změny výstupních proměnných během běhu smyčky na skutečných výstupech neprojeví.

Při psaní uživatelských programů je velmi důležité si tento princip uvědomit a jeho opomenutí může v určitých případech vést k problémům a zdánlivě špatné funkci programu.

3 Programování v SIMPLE 3

Jak již bylo řečeno, prvním krokem při tvorbě řídící aplikace je vytvoření zdrojového textu. Při tom je samozřejmě nutné držet se jistých striktních pravidel a zároveň je vhodné ctít i jisté zásady, které vytváření aplikace usnadní.

3.1 Uživatelský program a jeho zápis

Uživatelský program je tvořen, tak jak je běžné u programovacích jazyků, sledem deklarací, definicí, direktiv a příkazů jazyka, zakončený klíčovým slovem END. Textový soubor resp. soubory, obsahující program, nazýváme zdrojovým textem. Program není tvořen jen prostým sledem příkazů, které se transformují překladem na kód procesoru, ale jsou v něm i definice, deklarace a direktivy.

Definice ve zdrojovém textu jsou sděleními pro překladač, že nějaký objekt má námi určené vlastnosti. Chceme-li například konstantu 3.141592 pojmenovat Pi a dále toto jméno v programu používat, použijeme definici konstanty. Sdělíme tím překladači svůj požadavek. Ten si uloží jméno Pi a jemu přiřazenou hodnotu do svých tabulek, bezprostředně však nevytváří žádný kód pro procesor.

Deklarace jsou žádostí překladači o vytvoření místa v paměti pro objekt uvedených vlastností. Typicky se jedná o deklarace proměnných. V nich například sdělujeme překladači, že potřebujeme pole reálných proměnných o velikosti čtyři a chceme jej pojmenovat třeba Data. Překladač vymezí místo v paměti pro toto pole a uloží si do svých tabulek informaci, kde se pole Data nachází a že je to pole čtyř reálných proměnných. Opět se v tomto okamžiku nevytváří žádný kód pro procesor.

Direktivy jsou určeny pro řízení průběhu překladu. Sdělujeme jimi překladači, jakým způsobem má vytvářet výsledný kód, kdy přepnout na překlad z jiného souboru atd.

Pro příklad si uveďme jednoduchý program v jazyce SIMPLE3, ve kterém je deklarace jedné proměnné a jeden příkaz. K podrobnému vysvětlení syntaxe zápisů se pochopitelně dostaneme později.

Příklad jednoduchého programu v SIMPLE3

```
var i : word ;  end
i=i+1;
end
```

V tomto kratičkém programu je vytvořena proměnná **i** typu *word*. Ve druhém řádku programu je příkaz, který přiřazuje do proměnné **i** hodnotu, která se získá součtem původní hodnoty proměnné **i** a jedné. Jak již bylo řečeno dříve, je program vykonáván cyklicky, takže se nám s každým průchodem bude hodnota proměnné **i** zvyšovat o jednu.

3.2 Úprava textu

Ve snaze umožnit programátorovi přehlednou úpravu zdrojových textů, tvorbu dlouhých, samovysvětlujících identifikátorů a vhodné umísťování komentářů, jsou požadavky na úpravu zdrojového textu velmi volné. Jednotlivé příkazy, i jejich části mohou být na jednom řádku nebo „rozházeny“ na více řádcích. Počet mezer mezi klíčovými slovy, klíčovými znaky a identifikátory je neomezený. Konec řádku je vnímán překladačem jako znak mezery (s výjimkou řádkového komentáře). Jednotlivé příkazy mohou být od sebe oddělovány středníkem (jako v jazyce C nebo Pascalu). Překladač oddělovače příkazů nevyžaduje. Pro přehlednost je však doporučujeme.

1. Příklad volného uspořádání zdrojového textu

```
var i : word ;  end   i=i+1; end
```

2. Příklad volného uspořádání zdrojového textu

```
i         =
i     +
1;
```

3.3 Komentáře

Komentáře jsou dvojího druhu.

- ✓ Řádkový - uvozený dvojznakem **//** , ignoruje znaky až do konce řádku
- ✓ Strukturovaný - uvozený znakem **{** a ukončený znakem **}**. Strukturované komentáře je možné do sebe vnořovat.

Strukturovaný komentář se hodí zejména k „vystřihování“ částí zdrojového textu, který chceme dočasně potlačit. Komentář vnímá překladač jako mezeru.

Příklad řádkového komentáře

```
vari : word ;  end// deklarace proměnné i
i=i+1;            // přičítání jedničky
end               // konec programu
```

Příklad strukturovaného komentáře

```
vari,j,k : word ;  end
i=i+1;
{ dočasně vyjmuto
j=2*i;
k=3*j;}
end
```

3.4 Identifikátory

V programu pojmenováváme jednotlivé objekty, například proměnné, procedury a funkce, jmény, kterým souhrnně říkáme identifikátory. Identifikátor musí začínat písmenem, dále může obsahovat písmena, číslice a znak '_'. Počet znaků identifikátoru je omezen na 80.

Je-li identifikátor definován, je překladači znám až do konce překladu, s výjimkou formálních parametrů a lokálních proměnných procedur a funkcí, které jsou zapomenuty s ukončením definice procedury či funkce. Při definici nového identifikátoru překladač vyžaduje, aby nebyl shodný s některým již dříve definovaným identifikátorem, nebo předdefinovanou proměnnou.

Při porovnávání identifikátorů nejsou rozlišována malá a velká písmena.

Příklady správně vytvořených identifikátorů

```
  i
  x185
  DvereDoKravinaOtevreny
  Ventilator325
  Regulator_Spustit
```

Příklady nesprávně vytvořených identifikátorů

```
  7_Statecnych // první je číslice
x01234567890123456789012345678901234567890123456789012345678901234567890
1234567890123456789
               // příliš dlouhý identifikátor
VentilOtevren? // znak otazník není přípustný
```

Následující identifikátory v jednom řádku jsou považovány za shodné

```
  i       I
x185      X185
Ventil    ventil    VENTIL    vEnTiL
```

3.5 Typy

Každá proměnná, konstanta nebo funkce nese resp. vrací hodnotu, která zabírá nějaké místo v paměti a je nějak interpretována. Například na stejnou šestnáctici bitů se můžeme dívat jako na přirozené číslo v rozsahu {0,65535}, nebo na celé číslo v rozsahu {-32768 , 32767} atd. Souhrn těchto vlastností objektu nazýváme typ.

SIMPLE 3 rozlišuje čtyři základní datové typy. Jsou jimi BIT, WORD, INT a REAL. U typu BIT a WORD se rozeznává ještě varianta IN_BIT a IN_WORD, ze kterých lze informace pouze číst.

BIT	nese 1-bitovou logickou informaci o dvou možných hodnotách - pravda, nepravda - 1,0
IN_BIT	nese 1-bitovou logickou informaci o dvou možných hodnotách - pravda, nepravda - 1,0. Pouze ke čtení
WORD	nese 16-bitové kladné číslo v rozsahu {0,65535}
IN_WORD	nese 16-bitové kladné číslo v rozsahu {0,65535}. Pouze ke čtení.
INT	nese 16-bitové celé číslo v rozsahu {-32768 , 32767}
REAL	nese 32-bitové reálné číslo ve tvaru dle normy IEEE 754. Jedná se o čísla kódovaná v semilogaritmickém tvaru (znaménko)(mantisa)(znaménko exponentu)(exponent). Rozsah hodnot je {$-3.4x10^{38}$,...,$-1.5x10^{-45}$, 0 , $1.5x10^{-45}$,..., $3.4x10^{38}$} Přesnost čísla je 7-8 dekadických řádů

Zmíněné čtyři typy budeme nazývat <základní typy>.

Z proměnných základních typů lze vytvářet pole. Podle počtu dimenzí pole rozsahu indexů a typu prvků pak vzniká nepřeberné množství nových odvozených typů.

3.6 Konstanty

Prakticky v každém programu se vyskytují kromě proměnných také konstanty. Jsou to čísla, písmena atd., kterými plníme proměnné, přičítáme je, násobíme jimi apod. O způsobu zápisu těchto konstant v SIMPLE 3 pojednává tato kapitola.

3.6.1 Konstanty typu BIT

Konstanty typu BIT se v SIMPLE 3.0 nepoužívají. Přiřazení log 1 nebo log 0 se provádí speciálním příkazem nastavení či nulování. V logických výrazech použití konstant ztrácí smysl. Pokud se použití konstanty zdá výhodné, například z důvodů čitelnosti programu, je možné vyhradit dvě bitové proměnné, které na začátku programu inicializujeme.

3.6.2 Konstanty typu WORD a INT

Konstanty typu WORD a INT se zapisují stejně, jediný rozdíl je v tom, že znaménko mínus '-' může mít pouze INT. K zápisu hodnoty lze použít jednoho ze čtyř způsobů.

- ✓ Desítkové číslo
  sled decimálních cifer

- ✓ Hexadecimální číslo
  sled hexadecimálních cifer uvozených 0x

- ✓ Binární číslo
  sled binárních cifer uvozených 0b

- ✓ Znaková konstanta
  znak v apostrofech (hodnota je ASCII kód znaku)

Příklad zápisu konstant typu WORD:

```
  123     // desítkové číslo
0x2E3     // hexadecimální číslo
0b010011  // binární číslo
'A'       // znaková konstanta
-0xF3     // záporné hexadecimální číslo (INT)
-'0'      // záporně vzatý kód číslice 0
```

3.6.3 Konstanty typu REAL

Konstanty typu REAL zapisujeme obvyklým způsobem s použitím desetinné tečky a případného exponentu. Příklad je asi nejlepší:

Příklad zápisu reálných čísel:

```
  1.
-.0025
1.3E+3
-0.00000001E-20
1000000.
```

3.7 Definice konstant

V programech bývá účelné pojmenovávat konstanty jmény, která pak hodnotu konstanty zastupují. Usnadňuje se tak případná změna konstanty, která se pak provede na jediném místě. S výhodou se také použije pojmenovaných konstant na rozlišování stavů automatu, kódování prvků množin atd. Definice konstant má v SIMPLE3 následující podobu

Definice konstant je uvozena klíčovým slovem CONST. Následují definice jednotlivých identifikátorů. K přiřazení hodnoty identifikátoru je zvolen znak '#'. Na levé straně definice může být jak konstanta, tak i výraz, který lze vypočítat v době překladu. Mohou se v něm vyskytovat jak konstanty, tak identifikátory již definovaných konstant. Na pravé straně je nově definovaný identifikátor následovaný ukončovacím znakem ';'.

Zvláštní možnost je definice ENUM. Za tímto klíčovým slovem následuje seznam nově definovaných identifikátorů, oddělovaných čárkami. Jednotlivým identifikátorům jsou přiřazovány postupně hodnoty 0,1, atd. Definice je ukončena opět středníkem.

Celý blok definic je ukončen klíčovým slovem END.

Příklad definice konstant:

```
CONST
            123         #     CisloA ;     // Konstanta typu WORD
            3.141592    #     Pi ;         // Konstanta typu REAL
            2*Pi        #     DvePi ;
            ENUM   Po,Ut,ST,Ct,Pa,SO,Ne ;
            ENUM   Sever,Vychod,Jih,Zapad ;
END
```

3.8 Proměnné

SIMPLE3 používá soubor předdefinovaných proměnných, které jsou pevně umístěny v paměti a z nichž některé mají speciální funkce. Navíc si můžete příkazem var dodefinovat další proměnné, které se umísťují ve vlastním paměťovém prostoru. Tyto proměnné mohou být typu BIT,WORD,INTEGER nebo REAL, mohou z nich být vytvářena jedno i vícerozměrná pole. Navíc je umožněno více proměnným sdílet shodné paměťové místo, a tak ke stejnému místu přistupovat například jako k bitovému poli a WORDu zároveň. A teď již k definici podrobněji.

3.8.1 Vstupní, výstupní a předdefinované proměnné

Uživatelský program pracuje s proměnnými v paměti RAM, které v některých případech reprezentují fyzické vstupy a výstupy, které potom systémový firmware ošetří. K tomuto účelu je v paměti RAM automatu vyhrazena oblast, ve které jsou takto využívané proměnné umístěny. Z hlediska programování mají svá předdefinovaná jména (identifikátory), prostřednictvím kterých k nim v rámci aplikačního programu přistupujeme. Pokud systémový firmware nevyužívá danou proměnnou jako obraz fyzického vstupu, výstupu, nebo proměnné plnící speciální funkci, může být využita jako volná proměnná k potřebě programu.

identifikátor proměnné		typ	počet	rozsah	význam
X0	X255	IN_BIT	256	0 ... 1	číslicové vstupy
Y0	Y255	BIT	256	0 ... 1	číslicové výstupy
M0	M127	BIT	128	0 ... 1	„vnitřní relé“, (z toho M63-M127 jsou síťové)
B0	B127	BIT	128	0 ... 1	speciální funkční bity
I0	I63	WORD	64	0 ... 65535	analogové vstupy

O0	O63	WORD	64	0 ... 65535	analogové výstupy
D0	D63	WORD	64	0 ... 65535	obecné proměnné (z toho D32-D63 jsou síťové)
W0	W255	WORD	256	0 ... 65535	funkční registry
DX0	DX15	IN_WORD	16	0 ... 65535	Wordový přístup k X0 - X255
DY0	DY15	WORD	16	0 ... 65535	Wordový přístup k Y0 - Y255
DM0	DM7	WORD	8	0 ... 65535	Wordový přístup k M0 - M127
DB0	DB7	WORD	8	0 ... 65535	Wordový přístup k B0 - B127

Podle typu automatu se mění počet vstupních a výstupních proměnných využitých jako obraz fyzických vstupů a výstupů.

Proměnné, uvedené v tabulce můžeme používat, aniž bychom je museli deklarovat. Překladač o nich „ví“ od začátku překladu.

Některé z těchto proměnných, mající speciální funkci, mají definována ještě druhá jména. I tato jména jsou překladačem SIMPLE 3.0 rozeznávána.

3.8.2 Deklarace proměnných

Blok deklarací proměnných je uvozen klíčovým slovem VAR a ukončen klíčovým slovem END. Mezi nimi se nacházejí jednotlivé deklarace, které mohou být několika typů. Připomeňme, že hranaté závorky (pokud nejsou tučně) v popisu syntaxe vyjadřují volitelný prvek, složené libovolný počet opakování uzavřeného vzoru, tučně jsou vyznačena klíčová slova či znaky.

Typ proměnné je buď základní typ, nebo jedno či vícerozměrné pole, vytvořené na základě základní proměnné.

3.8.3 Nově deklarované proměnné

```
Syntaxe <jednotlivé deklarace proměnné>:
            <Nový identifikátor> {,<Nový identifikátor>} : <typ proměnné> ;
Syntaxe <typ proměnné>:
            array [<konstanta>{,<konstanta>}] of <základní typ>
            <základní typ>
```

Popis syntaxe říká, že napřed vyjmenujeme nově definované identifikátory nově definovaných proměnných, pak napíšeme dvojtečku a za ní uvedeme typ právě definovaných proměnných. Typ může být buď identifikátor základního typu, tedy BIT, WORD, INT nebo REAL, nebo může být odvozeným typem - polem. V případě pole píšeme klíčové slovo **array**, následuje výčet dimenzí v hranatých závorkách, potom následuje klíčové slovo **of** a základní typ, ze kterého je pole vytvořeno.

Deklarace proměnných - příklad

```
VAR   Flag1,Flag2       : BIT ;     // bitové
      i,j,k             : INT ;     // celočíselné se znaménkem
      aa,bb             : WORD ;    // celočíselné bez znaménka
      r1,r2,r3,r4       : REAL ;    // reálné
      pole              : ARRAY [5,12] OF INT ; // dvourozměrné pole
END
```

Konstanta v definici dimenzí pole může být i pochopitelně i identifikátorem konstanty. Konstanta určuje nejvyšší možný index na daném místě. Nejnižší index je 0. Indexujeme tedy počínaje nulou. Pole p[5] má tedy celkem 6 prvků a to p[0], p[1], p[2], p[3], p[4], p[5].

V případě vícerozměrného pole se index nejvíce vpravo „mění nejrychleji“. Tím chceme říci, že například prvky m[3,2] a m[3,3] leží v paměti za sebou.

Ukládání bitů a bitových polí se řídí následovně. Každá jednotlivá deklarace začíná na hranici celého bytu. Jednotlivé bity či bitová pole jsou umisťována v paměti postupně a to „bez mezer“.

Překročení meze pole

Překladač SIMPLE3 kontroluje překročení oblasti pole jak při překladu, pokud jsou indexy konstantní, tak i za běhu programu. Pokud by operace vedla k vybočení z oblasti paměti, vymezené pro dané pole, provede se operace s položkou pole s nejvyššími možnými indexy.

Reakce na překročení mezí pole - příklad

```
  pole[6,0]    // překročena oblast, pracuje se s prvkem
pole[5,12]
pole[0,15]// překročena mez indexu, ale není překročena oblast
pro pole.
          // Prvek je totožný s prvkem pole[1,2]
```

3.8.4 Synonyma proměnných

K již existující proměnné můžeme vytvořit synonymum, které může být stejného, nebo jiného typu. Lze tak jednu proměnnou zpřístupnit více jmény, nebo umožnit do stejného paměťového prostoru ukládání objektů různého typu. Hodí se takto například „wordový“ přístup k bitovým polím apod.

```
Syntaxe <vytvoření synonyma>:
     <identifikátor proměnné> # <Nový identifikátor>] [: <typ proměnné>] ;
```

Při kombinování přístupu WORD resp. INT a BIT je třeba mít na paměti, že proměnné WORD a INT jsou ukládány v paměti tak, že nižší byte leží na vyšší adrese, vyšší byte na nižší adrese. Bity a prvky bitových polí jsou ukládány vzestupně tak, že po sobě následují bity D0, D1, až D7 bytu, pak se přechází na D0 bytu s adresou o jedničku vyšší atd.

Deklarace proměnných - příklad

```
  VAR
     Flag1,Flag2     : BIT ;    // bitové
     i,j,k           : INT ;    // celočíselné se znaménkem
```

```
     aa,bb           : WORD ;  // celočíselné bez znaménka
     r1,r2,r3,r4     : REAL ;  // reálné
     aa   #    Baa   : ARRAY [15] OF BIT  ;// jiný přístup k aa
     aa   #    Xaa   : ARRAY [3,3] OF BIT ;// jiný přístup k aa
END
```

3.9 Výrazy

Jedním ze základních stavebních kamenů každého programovacího jazyka je výraz. Hodnoty výrazů ukládáme do proměnných, nebo jsou v případě hodnot typu BIT používány jako podmínky při větvení programů. Prvkem výrazu v SIMPLE 3.0 může být:

- ✓ Konstanta
- ✓ Proměnná, nebo prvek pole
- ✓ Volání standardní funkce
- ✓ Volání uživatelovy funkce
- ✓ Výraz v závorce

Každý prvek má svůj typ. Kontrolu slučitelnosti typů a případné konverze provádí kompilátor. Nemusíme se proto o převody starat. Je však vhodné mít o procesu konverze jasnou představu, protože se tak můžeme vyvarovat některým záhadným situacím.

Provedení operace mezi dvěma číselnými prvky (tj. typu INT, WORD nebo REAL) v případě rozdílnosti typů předchází konverze nižšího typu na vyšší. Pro tento účel je voleno následující uspořádání číselných typů:

```
INT < WORD < REAL
```

Konverze mezi INT a WORD nepředstavuje žádnou změnu bitové reprezentace objektu. Záporné číslo typu INT tak může způsobit překvapení zejména při porovnání . Číslo -1 konvertováno na WORD se totiž stává číslem 65535. Na tento efekt je třeba dát zvlášť dobrý pozor.

Konverze na typ REAL nepřináší žádná úskalí, neboť REAL umí pojmout čísla typu INT i WORD bez omezení. Při konverzi se provádí transformace bitové reprezentace na 32-bitový formát dle IEEE 754.

3.9.1 Operátory a jejich priorita

Prvky jsou ve výrazu vázány logickými operátory. Ty mohou být tzv. unární, vztahující se pouze k jednomu prvku, nebo binární, vážící dva sousední prvky. Operátory pro tvorbu výrazů jsou uvedeny v následující tabulce v pořadí jejich priorit

(nejvyšší je 0). Priorita určuje pořadí vykonávání operací v případě, že není určena závorkováním.

Název	operátor	výsledný typ	unární/binární	priorita
Logická negace	‘ (za prvkem)	BIT	unární	0
Unární mínus	- (před prvkem)	číselný	unární	0
Číselné dělení	/	číselný	binární	1
Číselné násobení	*	číselný	binární	1
Číselné sčítání	+	číselný	binární	2
Číselné odčítání	-	číselný	binární	2
Číselné menší	<	BIT	binární	3
Číselné menší nebo rovno	<=	BIT	binární	3
Číselné větší	>	BIT	binární	3
Číselné větší nebo rovno	>=	BIT	binární	3
Číselné rovno	=	BIT	binární	3
Číselné nerovno	<>	BIT	binární	3
Logické násobení	and	BIT	binární	4
Logický součet	or	BIT	binární	5
Exclusive or	xor	BIT	binární	5

Priority - příklad

```
  O1 + O2 * 123     // vypočte se nejprve součin, potom součet
X1' or O1<5 and O1>=3    // provede negaci X1, pak porovnání,
                         //pak and a nakonec or
```

3.9.2 Závorky

Jednotlivé části výrazu lze přirozeným způsobem závorkovat a tak buď zvýrazňovat, nebo měnit prioritu operací. Hloubka závorkování je omezena využitím paměti procesoru, které určité omezení hloubky přináší. Pokud však nepovažujete za omezení hloubku závorek cca 10, nenarazíte na žádnou překážku.

Výrazy - příklad

```
  O1 + 123                    // výraz typu WORD
O1 + 123.                     // výraz typu REAL
2*(O1+O2)/pole[i,j] + r1      // výraz typu REAL
X1' or (r1<5.) and (r1>=3.)   // výraz typu BIT
-(i+3*j)                      // výraz typu INT
```

3.9.3 Standardní funkce

K dispozici jsou následující standardní funkce

absolutní hodnota

(klíčové slovo abs) výsledný typ odpovídá typu skutečného parametru

celá část reálného čísla

(klíčové slovo trunc) výsledný typ je REAL

Skutečným parametrem funkcí je číselný výraz.

Volání standardní funkce - příklad

```
  ABS(i+O1)
trunc(r1/r2)
abs(abs(r1)-trunc(r1))
```

3.10 Definice symbolů

Definice symbolů nám umožňuje definovat identifikátor, který zastupuje úsek zdrojového textu, majícího význam výrazu. Při překladu je pak identifikátor symbolu nahrazován tímto úsekem.

Symbol oceníme zejména při označování vstupních a výstupních bitů, které mohou v průběhu vývoje aplikace změnit svou „logiku“. Změnu pak lze provést na jediném místě právě v definici symbolu. Stejně tak nám může symbol být prospěšný, pokud potřebujeme označit prvek uvnitř pole.

Blok definic je uvozen klíčovým slovem SYMBOL a ukončen klíčovým slovem END. Jednotlivé definice mají syntaxi:

```
Syntaxe <jednotlivé definice symbolu>:
            <výraz> # <Nový identifikátor> ;
```

Definice symbolů - příklad

```
  SYMBOL
     X1'        #    HorniHladina ;
     X2         #    DolniHladina ;
     HorniHladina' and DolniHladina'    #    Mezipoloha ;
     Y0'        #    StykacCerpadla ;
     Baa[5]     #    ChybaObsluhy ;
  END
```

3.11 Jednoduché příkazy

3.11.1 Přiřazení

Příkaz přiřazení umožňuje do proměnné, nebo prvku pole na levé straně příkazu přiřadit hodnotu výrazu na pravé straně příkazu. Hodnota výrazu je před uložením konvertována na typ proměnné.

Znakem přiřazení je =.

Na levé straně smí být i negace bitového objektu.

Příkaz přiřazení - příklad

```
i   = j + ABS(i+O1);
y1' = (r1<r2);
pole[3+j,k] = r1 + r3/3.;
O1  = pole[0,pole[1,i]+3]
```

3.11.2 Nastavení

Příkaz nastavení umožňuje definování hodnoty bitové proměnné, nebo negaci jejího obsahu.

- ✓ Nastavení na 1 provedeme samostatným uvedením identifikátoru bitové proměnné, nebo prvku bitového pole.
- ✓ Nastavení na 0 provedeme uvedením znaku negace ‘ za identifikátorem proměnné nebo prvku bitového pole.
- ✓ Negace se provede uvedením znaku vykřičník ! před identifikátorem proměnné nebo prvku bitového pole.

Příkaz nastavení - příklad

```
Y0;      // nastavení Y0 na hodnotu 1
Y0';     // nastavení Y0 na hodnotu 0
!Y0;     // negace hodnoty Y0
Baa[i]';// nastavení prvku bitového pole na 0
```

3.11.3 Volání procedury

Za jednoduchý příkaz je považováno i volání procedury (subroutine). Na rozdíl od předchozí verze SIMPLE může mít procedura i funkce v jazyce SIMPLE 3.0 formální parametry. Více je o volání procedur a funkcí v kapitole „Procedury a funkce“.

3.11.4 Zobrazení

Příkaz zobrazení se svým zápisem podobá příkazu přiřazení. Na levé straně se však nachází fiktivní proměnná DISPLAY, na pravé je buď výraz nebo řetězec v uvozovkách. Zobrazení čísla se řídí ještě obsahem proměnné FORMAT.

Příkaz zobrazení - příklad

```
display = O1;
display = "Doba uderu-derovac 1";
```

3.12 Programové struktury

Programové struktury jsou jazykové konstrukce, které nám slouží ke větvení programu na základě hodnot výrazů. V běžných programovacích jazycích jsou používány struktury jak pro větvení, tak i pro vytváření cyklů. Cykly jsou však v SIMPLE 3.0 záměrně vynechány, aby nemohl uživatel v žádném případě vytvořit program, který by nikdy nepředal řízení zpět systémové vrstvě a tak zcela „zatuhl“.

3.12.1 Struktura IF

Pro větvení programu na základě pravdivosti logického výrazu slouží struktura **IF**.

```
Syntaxe <struktura IF>:
      if    <Logický výraz >
      then  <blok>
      {     elseif <Logický výraz >       then   <blok> }
      [     else   <blok>     ]
      endif
```

Slovem blok rozumějme sled jednoduchých příkazů nebo programových struktur.

Jak je patrno ze syntaxe, uvozuje strukturu klíčové slovo **if**. Za ním je logický výraz. za ním slovo **then**. Následuje blok. Ve struktuře může být několik větví **elseif** a nejvýše jedna větev **else**. Struktura je uzavřena klíčovým slovem **endif**.

Vysvětleme si význam struktury nejdříve na nejjednodušším případu, kdy nejsou větve elseif a else použity.

IF-THEN-ENDIF - příklad

```
IF  x1 THEN  y1; y3;  ENDIF
```

V uvedeném příkladu program otestuje hodnotu x1. Je-li hodnota rovna log.1, provede se sled příkazů za then, tedy nastavení y1 a y3, a pokračuje za slovem endif. Pokud je hodnota x1 rovna log.0, přeskočí program blok za then a pokračuje rovnou za slovem endif.

Přidáním větve **else** získáme strukturu, která umožňuje rozvětvit program na dvě cesty.

IF-THEN-ELSE-ENDIF - příklad

```
IF  x1
 THEN  y1; y3;
 ELSE  y2; y4;
ENDIF
```

V uvedeném příkladu program otestuje hodnotu x1. Je-li hodnota rovna log.1, provede se sled příkazů za then , tedy nastavení y1 a y3, a pokračuje za slovem

endif. Pokud je hodnota x1 rovna log.0, přeskočí program blok za then a pokračuje za slovem else, provede tedy nastavení y2 a y4. Potom pokračuje za slovem endif.

Přidáním větví **elseif** získáme strukturu, která umožňuje rozvětvit program na více cest.

IF-THEN-ELSE-ENDIF - příklad

```
  IF      x1  and x2  THEN  y1; y3;
ELSEIF  x1  and x2' THEN  y2; y4;
ELSE                      y1; y4;
ENDIF
```

V uvedeném příkladu program otestuje hodnotu logického výrazu (x1 and x2). Je-li hodnota rovna log.1, provede se sled příkazů za then , tedy nastavení y1 a y3, a pokračuje za slovem endif. Pokud je hodnota rovna log.0, vyhodnocuje se hodnota logického výrazu (x1 and x2'). Je-li tato hodnota rovna log.1, provede se sled příkazů za then , tedy nastavení y2 a y4, a pokračuje za slovem endif. Pokud je hodnota výrazu rovna log.0, přeskočí program blok za then a pokračuje za slovem else, provede tedy nastavení y1 a y4. Potom pokračuje za slovem endif.

Příklad vnořených struktur IF

```
  IF     Vst1 THEN y1; y2;
  ELSEIF Vst2 THEN IF   Pocet>4 THEN DISPLAY='*';
                   ELSE             DISPLAY=' ';
                   ENDIF
                   !y3; !y2;
  ELSE              y2'
  ENDIF
```

3.12.2 Struktura CASE

Další programovou strukturou je struktura **CASE-OF-ELSE-ENDCASE**. Slouží k rozskokům dle hodnoty celočíselného výrazu.

```
Syntaxe <struktura CASE>:
      case  <Celočíselný výraz >
      {     of    <Konstantní celočíselný výraz > then <blok> }
      [     else  <blok>      ]
      endcase
```

Jednotlivé alternativy za klíčovými slovy **OF** musejí být celočíselné konstantní výrazy, tj. hodnoty, vypočitatelné již při překladu. Alternativa **ELSE** je nepovinná.

Struktura CASE je vlastně zjednodušeným zápisem speciálního případu struktury IF. Hodnota celočíselného výrazu za case se postupně porovnává s konstantními hodnotami za slovy of, dokud nenastane shoda. Pokud nastane, provede se blok za then a pokračuje se za slovem endcase. Zbylé konstanty se již neporovnávají. Nenastane-li shoda ani v jednom případě, provede se blok za slovem else, pokud existuje. Potom se pokračuje za slovem endcase.

CASE-OF-THEN-ELSE-ENDCASE - příklad

```
CASE den
 OF     1    THEN  DISPLAY="Po"
 OF     2    THEN  DISPLAY="Ut"
 OF     3    THEN  DISPLAY="St"
 OF     4    THEN  DISPLAY="Ct"
 OF     5    THEN  DISPLAY="Pa"
 OF     6    THEN  DISPLAY="So"
 OF     7    THEN  DISPLAY="Ne"
ELSE              DISPLAY="??"
ENDCASE
```

3.13 Procedury a funkce

Procedury a funkce jsou části programu, které je možné pojmenovat identifikátorem a poté tento identifikátor použít jako příkaz programu. Výsledkem uvedení identifikátoru funkce nebo procedury je provedení činnosti, kterou jsme dříve tímto identifikátorem pojmenovali.

Překladač v místě uvedení identifikátoru procedury či funkce neumísťuje znovu a znovu celý sled operací, který identifikátor reprezentuje, ale umísťuje pouze instrukci volání, která přesměruje chod programu do takzvaného těla procedury. Sekvence operací procedury či funkce tak existuje v paměti programu pouze jednou. Samotné volání zabírá minimální prostor v paměti.

Používání procedur a funkcí je nejen prostředkem k šetření paměti programu, ale také, ne-li především, k zpřehledňování textu programu a zvyšování jeho čitelnosti.

Procedury i funkce mohou mít formální parametry, i lokální proměnné. Vysvětleme si co to znamená.

3.13.1 Formální parametry

Formální parametry jsou při definici procedury nebo funkce uvedeny v závorce za jejím identifikátorem.

Formální parametry - příklad

```
  SUBROUTINE   ZobrazDen( den,kam:WORD )
PROCEDURE ZvysOJednaModulo(  var co : WORD ; mod : WORD )
```

Budeme-li chápat; proceduru jako předpis k provedení určité složitější operace, představuje formální parametr (parametry) objekt, se kterým, nebo s jehož pomocí, se má operace provést. Příkladem může být procedura, která zobrazuje námi definovaným způsobem den v týdnu, který máme číselně kódován. Číslo dne předáme proceduře jako parametr.

Při předávání parametru můžeme mít rozdílné požadavky. U zmíněného zobrazení dne stačí proceduře sdělit hodnotu. Pokud však potřebujeme, aby procedura „opracovala“ objekt, předání hodnoty nestačí. Chceme-li zvýšit hodnotu proměnné o jedna modulo základ, musíme proceduře zpřístupnit proměnnou celou, nejen „pro čtení“ , ale i pro „zápis“. Musíme proceduře sdělit, kde leží. Proto jsou procedurám a funkcím umožněny dva způsoby přístupu k formálním parametrům:

Předání hodnotou

Při předání hodnotou je formální parametr představován vnitřní proměnnou procedury, do které si procedura uloží předávanou hodnotu. Pokud se v proceduře objeví identifikátor formálního parametru, pracuje se s takto vytvořenou proměnnou. Skutečným parametrem, předaným proceduře, pak může být i výraz.

Předání odkazem

Při předání odkazem je identifikátor formálního parametru zástupným jménem pro proměnnou, která bude skutečným parametrem při volání. Všechny operace procedury s formálním parametrem se při zavolání provedou přímo s proměnnou, která je proceduře předána. Skutečným parametrem smí být pouze proměnná nebo prvek pole.

Definice formálních parametrů

Definice formálních parametrů je uváděna za identifikátorem nově definované procedury nebo funkce. Je uzavřena v kulatých závorkách.

```
Syntaxe <definice formálních parametrů>:
      ({[var]<Nový identifikátor>{,<Nový identifikátor>}:<základní typ>;})
```

Klíčové slovo var předznamenává parametry předávané odkazem, ostatní jsou předávány hodnotou.

Formální parametry - příklad

```
  SUBROUTINE   Cosi( var a,b : REAL ; ii, jj : INT ;  bb : BIT
)
          // a,b    jsou parametry typu REAL předávané odkazem
          // ii,jj  jsou   parametry   typu   INT     předávané
hodnotou
          // bb     je parametr typu BIT předávaný hodnotou
```

Definice procedury

Definice procedury je uvozena klíčovým slovem PROCEDURE nebo SUBROUTINE. Ukončena je klíčovým slovem RETURN.

```
Syntaxe <definice procedury>:
      PROCEDURE <Nový identifikátor><definice formálních parametrů>
            { [<deklarace lokálních proměnných>]
              [<definice symbolů>]
              [<definice konstant>] }
            { <blok> }
      RETURN
```

Za definicí identifikátoru procedury a formálních parametrů může následovat deklarace lokálních proměnných, konstant a symbolů. Lokální proměnné jsou proměnné, které procedura používá pro svou činnost a jsou nepřístupné mimo právě definovanou proceduru. Mohou sloužit například jako pomocné. S ukončením definice procedury jsou identifikátory formálních parametrů i lokálních proměnných překladačem zapomenuty, a lze je tedy v programu použít znovu. Jsou jim pak přiřazeny zcela nové objekty.

V SIMPLE3 získávají lokální proměnné pevné místo v paměti. Procedura tedy najde jejich obsah při příštím zavolání ve stavu, v jakém je naposledy zanechala. Toho lze v programech s výhodou využívat k sumacím, uchovávání stavové informace apod.

Samotné tělo procedury je sledem jednoduchých příkazů nebo programových struktur. Zdůrazněme, že jména všech globálních objektů, jako jsou proměnné, symboly a konstanty, definované mimo těla procedur a funkcí, jsou normálně přístupná. Nejsme tedy omezeni pouze na lokální objekty.

Definice procedury - příklad

```
  PROCEDURE PrictiJednaModulo( var kam : word ; mod : word )
     kam = kam + 1;
      IF kam>=mod THEN kam = 0 ENDIF
 RETURN
```

Definice procedury - příklad

```
  SUBROUTINE ZobrazDen( den,kam:WORD )
     POSITION = kam;
     CASE den
     OF   1    THEN  DISPLAY="Po"
     OF   2    THEN  DISPLAY="Ut"
     OF   3    THEN  DISPLAY="St"
     OF   4    THEN  DISPLAY="Ct"
     OF   5    THEN  DISPLAY="Pa"
     OF   6    THEN  DISPLAY="So"
     OF   7    THEN  DISPLAY="Ne"
     ELSE            DISPLAY="??"
     ENDCASE
RETURN
```

3.13.2 Volání procedury

Volání procedury se provede uvedením jejího identifikátoru a předáním skutečných parametrů. Skutečné parametry jsou uvedeny v závorce a mezi sebou odděleny čárkami. Pořadí skutečných parametrů odpovídá pořadí formálních parametrů. Musí jich být stejný počet jako je formálních parametrů a musejí mít kompatibilní typ. Číselné parametry předávané hodnotou jsou kompatibilní v tom smyslu, že kompilátor hodnotu před předáním v případě rozdílnosti typů převede na typ požadovaný. Konverze mezi číslem a parametrem BIT však není možná, stejně tak musejí být shodné typy u parametrů předávaných odkazem.

Příklad definice procedury

```
  PROCEDURE    Cosi( var a,b : REAL ; ii, jj : INT ;  bb : BIT
)
          // a,b    jsou parametry typu REAL předávané odkazem
          // ii,jj  jsou   parametry   typu   INT     předávané
hodnotou
          // bb     je parametr typu BIT předávaný hodnotou
```

Příklad volání procedury

```
Cosi( r1,r2,(i+j)/3,15,O2<100 );    // správné volání procedury
```

Příklad chybného volání procedury

```
Cosi( 3.14,r2,i,j,x0 ); // chyba - první parametr je třeba předat odkazem,
                              // musí tedy být identifikátor proměnné

Cosi( r1,r2,i,j );      // chybný počet parametrů

Cosi( r1,r2,Y0,r3,x0 ); // chyba - třetí parametr neodpovídá typem
```

3.13.3 Definice funkce

Funkce se svou definicí příliš neliší od procedur. Na rozdíl od procedur však vracejí hodnotu. Proto je jim přiřazen typ. Hodnota funkce se předává přiřazovacím příkazem, na jehož levé straně je identifikátor definované funkce.

```
Syntaxe <definice funkce>:
      FUNCTION <Nový identifikátor><definice formálních parametrů>
            : <základní typ>;
            { [<deklarace lokálních proměnných>]
              [<definice symbolů>]  [<definice konstant>] }
            { <blok> }
      RETURN
```

Definice funkce - příklad

```
  FUNCTION RegulacePI( VAR Sum :REAL; e,p,i :REAL ) : REAL ;
     Sum = Sum+e ;
     RegulacePI = p*e+i*Sum;
RETURN
```

Tělo programu

Každý příkaz, definovaný mimo proceduru či funkci, je příkazem základní programové smyčky. I když je tak umožněno vzájemné prolínaní textu základního programu a procedur či funkcí, nelze takovýto programátorský styl doporučit.

Za vhodné považujeme nejdříve definovat všechny procedury a funkce, teprve nakonec definovat tělo programu. Není na škodu komentářem zdůraznit začátek. Konec programu je vyznačen povinným klíčovým slovem END.

3.14 Direktivy

Direktivy modifikují způsob překladu zdrojového textu. Mezi direktivy SIMPLE 3.0 patří příkaz #INCLUDE, NetAddr, #pragma $LJ a #pragma $SJ.

3.14.1 #INCLUDE

Direktiva #INCLUDE umožňuje přepnout překlad do jiného souboru, a po jeho skončení se vrátit do původního. Direktivu #INCLUDE je možné použít na libovolném počtu úrovní. Toto členění zdrojového textu umožňuje sdílet některou část více programy a udržovat ji tak jen v jednom exempláři. Obsahem include souboru může být jedna nebo více definic konstant, symbolů, proměnných, procedur a funkcí.

```
Syntaxe <include>
      #INCLUDE "<soubor>" ;
```

3.14.2 NetAddr

Direktiva NetAddr definuje adresu stanice v síti PESNET, pro kterou je daný program určen.

```
Syntaxe <NetAddr>
      NetAddr (<konstanta typu WORD>);
```

3.14.3 Pragma

Při generování kódu používá překladač pro větvení programu podmíněné skoky. Ty jsou u procesoru řady 8051 omezeny délkou skoku o 127 bytů. Instrukce těchto skoků zabírají málo místa v paměti a jsou rychlé, nestačí však vždy našim požadavkům. Pokud je větší počet příkazů za klíčovým slovem THEN, nebo používáme vnořené struktury, brzy na hranici 127 bytů narazíme. Direktiva #pragma $LJ;

přepíná překladač do režimu, ve kterém tyto krátké skoky nahrazuje dvojicí krátkého podmíněného a dlouhého nepodmíněného skoku. Tato kombinace zabírá o něco více místa v paměti, o pár taktů procesoru prodlužuje vykonání programu, avšak umožňuje nám se vyjádřit tak, jak jsme chtěli. Direktiva #pragma $SJ; přepíná překladač zpět do původního režimu. Máme tak možnost určit, ve které části programu má být použit který způsob překladu.

4 Speciální funkce

Mimo proměnných, reprezentujících fyzické vstupy a výstupy má celá řada předdefinovaných proměnných speciální význam. Zápisem do nich můžeme ovlivňovat funkci terminálu, řídit jeho činnost nebo se dovídat o jeho stavu. Jsou to vnitřní časovače, reálný čas, informace o rychlosti systému atd. Pro realizaci speciálních funkcí se využívají některé bity ze sady B0 - B127 a některé registry ze sady W0 - W127. Všechny proměnné, které mají speciální funkce, mají napevno přiřazen ještě jeden identifikátor, kterým se symbolicky vyjadřuje jeho funkce. Doporučujeme všude v programu používat tyto názvy - klesá možnost omylů. Následující tabulka dává jejich základní přehled.

I když většina bitů B a registrů W je zatím neobsazena, není vhodné je v programu využívat jako obecné proměnné (z důvodů kompatibility s dalšími verzemi jazyka SIMLE do budoucna).

Identifikátor	Umístění	Popis funkce
RESET	B126	počáteční inicializace systému
T0-T7	W0-W7	šestnáctibitové časovače
TEN0-TEN7	B0-B7	povolení čítání
TDM0-TDM7	B8-B15	čítání dolů
TPA0-TPA7	B16-B23	předdělič stem
TOE0-TOE7	B24-B31	čítání přes přetečení
TOF0-TOF7	B32-B39	příznak přetečení
CLK0-CLK7	B40-B47	asynchronní impuls na časovač
SPEED	W15	rychlost systému
CALIB0-7	W18-W25	kalibrace analogových vstupů
ADCMODE0-7	W26-W33	nastavení modu A/D (jen u vybraných typů)
SECOND	W8	sekundy reálného času. (1..59)
MINUTE	W9	minuty reálného času (0-59)
HOUR	W10	hodiny reálného času (0-23)
DAY	W11	dny v měsíci (1-28..31)
MONTH	W12	kalendářní měsíc (1-12)
YEAR	W13	roky století (0-99)
WEEK	W14	dny v týdnu (1-7)
HOLD	B48	vzorkování reálného času
POSITION	W34	pozice kursoru
FORMAT	W35	formát zobrazení znaku
KBCODE	W36	kód stisknuté klávesy
KBDELAY	W37	prodleva "autorepetu"
KBREPEAT	W38	rychlost "autorepetu"
CRLSEC	B49	zaokrouhlení na minuty
KBREPEN	B50	povolení autorepetu
KBSOUND	B51	aktivace zvukové odezvy klávesnice
LOADGEN	B52	aktivace uživatelských znaků
FASTADC	B53	nastavení modu rychlého AD převodu
EESAVE	B54	aktivace zápisu proměnných do EEPROM
EELOAD	B55	aktivace vyčtení proměnných z EEPROM
EEWR_COUNT	W39	čítač zápisů do EEPROM
TURBO	B56	přepínač pro dvojnásobnou rychlost systému
SENDCODE	W42	kód instrukce pro síťovou operaci
SENDADDR	W43	adresa stanice vybrané pro síťovou komunikaci
MEMADDR	W44	adresa v paměti vybrané stanice
BITADDR	W45	bitová adresa v paměti vybrané stanice
SENDDATA	W46	data přijímaná ze sítě či odesílaná do sítě
SENDBIT	B57	datový bit přijímaný ze sítě či odesílaný do sítě
MYADDR	W50	vlastní adresa stanice

4.1 Reset

RESET je speciální funkční bit, který je automaticky nastaven vždy po zapnutí automatu a též vždy po spuštění programu. Tento bit lze využít v programu na počáteční inicializace a pak jej vynulovat, nebo se může použít i jinak, je však třeba počítat s tím, že po každém restartu automatu se tento bit nastaví.

Symbol	Umístění	Popis funkce
RESET	B126	počáteční inicializace systému

4.2 Rychlost sytému

V automatu je průběžně aktualizována proměnná SPEED, ve které je uveden počet průchodů celým programem za sekundu. V aplikacích náročných na rychlost odezvy lze například tento registr testovat a kontrolovat, zda se regulační program nedostává do časového skluzu.

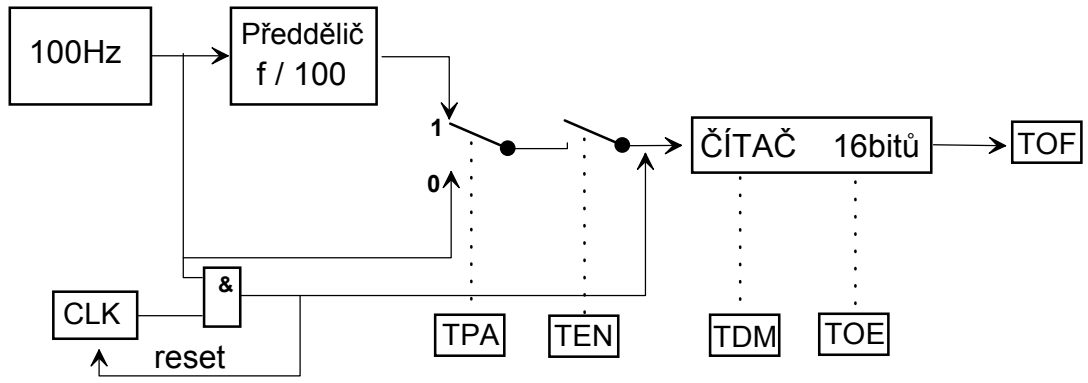
Symbol	Umístění	Popis funkce
SPEED	W15	rychlost systému

4.3 Časovače

Časovače jsou reprezentovány specielními funkčními registry (WORD) se symbolickými názvy T0 až T7. Všech osm čítačů je zcela nezávislých a se shodnými funkcemi.

Činnost každého časovače souvisí se samostatnou šesticí funkčních bitů TENn, TPAn, CLKn,TOEn, TOFn, TDMn, kde n=0-7 specifikuje příslušný čítač. Funkci těchto bitů naznačuje obrázek.

Časovač T a jeho řídící bity:



Symbol	Umístění	Popis funkce
T0-T7	W0-W7	šestnáctibitové časovače
TEN0-TEN7	B0-B7	povolení čítání
TDM0-TDM7	B8-B15	čítání dolů
TPA0-TPA7	B16-B23	předdělič stem
TOE0-TOE7	B24-B31	čítání přes přetečení
TOF0-TOF7	B32-B39	příznak přetečení
CLK0-CLK7	B40-B47	asynchronní impuls na časovač

TEN povolení čítání časovače
TEN0=0 časovač T0 stojí
TEN0=1 časovač T0 čítá

TPA zařazuje předdělič 1/100
TPA0=0 časovač T0 čítá po 10 ms
TPA0=1 časovač T0 čítá po 1 s

CLK vygeneruje 1 impuls na časovač (nezávisle na TEN) a ihned se vynuluje. Tímto způsobem lze asynchronně přidávat impulsy na časovač.

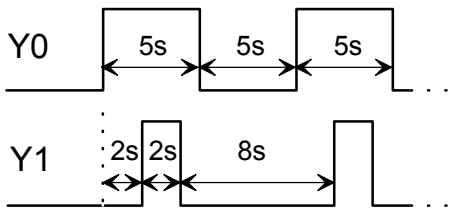
TOE povoluje čítání přes přetečení
TOE0=0 časovač T0 zastaví na 65535 (čítá-li nahoru), nebo na 0 (čítá-li dolů)
TOE0=1 časovač T0 po dosažení meze nezastavuje, ale přetéká (při čítání nahoru 65535->0, při čítání dolů 0->65535)

TOF informace o přetečení časovače (musí být nulován programem)

TDM směr čítání časovače
TDM0=0 časovač T0 čítá nahoru
TDM0=1 časovač T0 čítá dolů

Příklad: ovládání výstupů Y0 a Y1 podle časových průběhů na obrázku

```
  IF RESET THEN; Y0´; Y1´;
     TEN0;  TPA0;  TDM0´;  TOE0´;
T0=0;
     RESET´;
ELSE
     IF    (T0=> 5) THEN Y0;
     ELSEIF(T0=> 7) THEN Y1;
     ELSEIF(T0=> 9) THEN Y1´;
     ELSEIF(T0=>10) THEN Y0´; Y1´;T0=0;
     ENDIF
ENDIF
END
```



4.4 Reálný čas

V jazyce SIMPLE3 můžou být programovány automaty, mající prostředky pro práci s reálným časem. V tomto případě jsou uživateli přístupné ve speciálních funkčních registrech údaje o sekundách, minutách a hodinách reálného času. Kromě toho jsou k dispozici i registry s informací o dnech v měsíci, o měsících, letopočtu a dnech v týdnu. Hodinové údaje jsou ve 24 hodinových cyklech. Kalendář nectí přestupné roky. Dny v týdnu nejsou závislé na datu, čili se jedná o nezávislý čítač modulo 7 (1až7). Je tedy na uživateli, kterým dnem bude začínat týden. Provede-li se však nastavení podle systémového času osobního počítače pomocí programu LOADER.EXE nebo SETPES.EXE, potom týden začíná nedělí. Funkční bit HOLD slouží ke vzorkování reálného času. Po nastavení tohoto bitu se údaje v časových registrech nemění, čas však nadále běží na pozadí. Po vynulování tohoto bitu se opět v registrech objeví údaje odpovídající reálnému času. Nastavením bitu CLRSEC se zaokrouhluje reálný čas na celé minuty. Je vynulován obsah sekundového registru, a pokud je před nulováním obsah sekundového registru větší než 29, je zároveň inkrementován obsah minutového registru. Tento funkční bit je po zaokrouhlení času automaticky nulován. Proměnné vyhrazené pro reálný čas jsou:

Symbol	Umístění	Rozsah	Popis funkce
SECOND	W8	0 - 59	sekundy reálného času. (1..59)
MINUTE	W9	0 - 59	minuty reálného času (0-59)
HOUR	W10	0 - 23	hodiny reálného času (0-23)
DAY	W11	1 - 31	dny v měsíci (1-28..31)
MONTH	W12	1 - 12	kalendářní měsíc (1-12)
YEAR	W13	0 - 99	roky století (0-99)
WEEK	W14	1 - 7	dny v týdnu (1-7)
HOLD	B48	0	registry reálného času neustále aktualizovány
		1	registry reálného času „drží“ poslední hodnotu
CRLSEC	B49	0 - 1	zaokrouhlení na minuty

Pozn.: Při pokusu o využití služeb reálného času na automatech, kde není obvod reálného času osazen, je třeba počítat s tím, že v registrech budou nesmyslné hodnoty.

Příklad použití reálného času:

```
  {----------------------Deklarace--------------------------}
VAR
     XO # Cidlo1;         // Infra čidlo uhlavního vchodu
     X1 # Cidlo2;         // Infra čidlo u zadního vchodu
     Y0 # Ochranka;       // signalizace pro ochranku
     M0 # VolnyPruchod;
     X2 # Tlacitko;       // Kvitovaci tlacitko
END
SYMBOL
     VolnzPruchod' #  ZakazanyPruchod;
     (Week=7)      #  Sobota;
     (Week=1)      #  Nedele;
     ((Hour=>5) and (Hour<=17))    # Pracovní doba;
     ((Day =28) and (Month=10))    # Státní svátek;
END
{----------------------Program----------------------------}
```

```
ZakazanyPruchod = (Sobota or Nedele or Statni svatek or
Pracovni doba´)

IF   VolnyPruchod' and (Cidlo1 or Cidlo2)
THEN Ochranka; // Zapnuti alarmu
ENDIF

IF   Tlacitko
THEN Ochranka´;// Vypnuti alarmu
ENDIF

END
```

4.5 A/D převodník

U automatů s analogovými vstupy se hodnoty těchto vstupů objevují v proměnných I0, I1 ...atd.

Rozsah těchto hodnot může být obecně 0.....65535. Jednotky, ve kterých je daná veličina měřená a její maximální rozsah, je dán konkrétním typem analogového vstupu. Například typ A3 měří v jednotkách 0,01 mA, typ A6 může měřit teplotu v jednotkách 0,1 °K nebo odpor v jednotkách 0,01Ω apod.

4.5.1 Rychlost A/D převodu

U standardních analogových vstupů (vyjma vstupů pro odporová čidla) je prováděna automatická digitální filtrace. Krok jednoho měření přes všechny vstupy trvá 350 ms. Aktuální hodnoty jsou do proměnných I0..In přepsány vždy na začátku programové smyčky a v celém průběhu programu smyčkou se nemění. V případě, že je potřeba zrychlit měření je to možné nastavením speciálního funkčního bitu FASTADC, v tomto případě je krok měření 50 ms. Zrychlení se samozřejmě děje na úkor přesnosti měření.

4.5.2 Kalibrace analogových vstupů

Kalibrace analogových vstupů se provádí pomocí speciálních funkčních proměnných CALIB0 až CALIB7 (max. počet podporovaných analogových vstupů je 8) Po resetu programu je v těchto proměnných vždy hodnota 10 000. Tato hodnota je chápána jako 1,0000 - a může být měněna uživatelským programem od 0,0 do 1,6000 (zápisem hodnot 0..16000 do proměnných CALIB0 až CALIB7)

Tato hodnota funguje jako násobná pro každý kanál, lze tedy chápat převod každého kanálu In takto:

In= (nominální hodnota analog. vstupu) * (CALIBn/10000)

Máme čidlo, připojené na vstup I2 automatu AnneX, které má výstup 0-20 mA a chceme výslednou veličinu na vstupu I2 vidět přímo v procentech (tedy 0-100). Standardně se kanál I2 zobrazuje v 0.01mA, viděli bychom tedy hodnoty od 0 do 2000. Pokud tedy po resetu přestavíme CALIB2 na hodnotu 500, bude se výsledná hodnota násobit koeficientem 0.500 a v I2 nyní uvidíme hodnoty 0..100.

Příklad zápisu v programu:

```
  VAR
     Y0 # DOLNI_POLOVINA;
     Y1 # HORNI_POLOVINA;
END
IF RESET THEN CALIB2=500; RESET'; ENDIF
DOLNI_POLOVINA = I2 < 50;
HORNI_POLOVINA = I2 > 50;
END
```

4.5.3 Módy funkce analogových vstupů pro měření odporu

U automatů a řídících stanic vybavených analogovými vstupy pro přímé připojení odporových čidel je možné zvolit mód činnosti respektující způsob připojení čidla a měřící rozsah. Uvedený mód činnosti se volí nastavením funkční proměnné ADCMODE.

Možnosti nastavení módu činnosti analogových vstupů

ADCMODE	Měřící rozsah	Zobrazovaná hodnota	Připojení čidel
0	500Ω	odpor s rozlišením 0.01Ω	Třívodičově
4	5kΩ	odpor s rozlišením 0.1Ω	
8	500Ω	teplota s rozlišením 0.1K	
16	500Ω	odpor s rozlišením 0.01Ω	Dvouvodičově
20	5kΩ	odpor s rozlišením 0.1Ω	
24	500Ω	teplota s rozlišením 0.1K	

Funkčí proměnné pro ovládání A/D převodu

Symbol	Umístění	Popis funkce
CALIB0-7	W18-W25	kalibrace analogových vstupů
ADCMODE	W26	nastavení modu A/D (jen u vybraných typů)
FASTADC	B53	nastavení modu rychlého AD převodu

4.6 Ukládání dat do paměti EEPROM

V jistých případech je výhodné zálohovat určitý malý objem “důležitých” dat zápisem do paměti EEPROM. Tento způsob zálohování je přínosný především u takových konfigurací automatů, které nejsou vybaveny datovou pamětí typu NVRAM. Zápis do EEPROM je nefunkční, jestliže je programová paměť uzamčena.

Nastavení funkčního bitu EESAVE dojde k přepisu právě aktuelního obsahu proměnných D0 až D29 a M0 až M31 do zálohové paměti EEPROM. V případě, že tato uložená data chceme vyčíst, provedeme nastavení funkčního bitu EELOAD. Oblast proměnných D0 až D19 a M0 až M31 se naplní daty z EEPROM. Je třeba zdůraznit, že zápis dat do EEPROM je nutné provádět pouze při splnění nadefinované podmínky, a to proto, aby se zápis neprováděl vždy při kadém průchodu tělem programu. Pamět typu EEPROM má totiž omezený počet zápisů (50 000). Proto je toto zálohování třeba provádět uváženě. Počet již provedených zápisů je možno zjistit vyčtením hodnoty specielní funkční proměnné EEWR_COUNT.

Symbol	Umístění	Popis funkce
EESAVE	B54	zápis proměnných D0 - D29,M0 - M31 do EEPROM
EELOAD	B55	čtení proměnných z EEPROM do D0-D29,M0-M31
EEWR_COUNT	W39	udává počet již provedených zápisů do EEPROM

Příklad:

```
    IF RESET THEN LOAD;RESET¨'; ENDIF

    {Naplní po startu oblasti proměnných D0 - D29 a M0 - M31
dříve uloženými daty.}



    IF KBCODE=8 then SAVE; ENDIF

    {Klávesou F1 zapiseme oblasti promenných D0 - D29 a M0 -
M31 do zalohovaci pameti.}

    IF KBCODE=9 then POSITION=0; DISPALY=CITAC;  ENDIF

    {Klávesou F2 vypiseme na display pocet provedenych zapisu
do zalohovaci pameti}
```

5 Terminálové funkce

Programovatelné terminály řady “T” a kompaktní řídící stanice řady “PP” vyráběné firmou HYPEL jsou shodně vybaveny klávesnicí a displayem. Na tyto periferie musí samozřejmě vhodně přistupovat uživatelským programem a k tomuto má jazyk SIMPLE3 příslušné nástroje.

5.1 Klávesnice

Všechny programovatelné terminály řady “T” a kompaktní řídící stanice řady “PP” mají membránovou klávesnici s 21 tlačítky s mechanickou odezvou. Vstupní informace z klávesnice je kódována do funkčního registru KBCODE tak, že každému tlačítku odpovídá určitý číselný kód. Pokud je stisknuto více kláves současně, je v proměnné KBCODE číselný kód jen jedné z nich. Tento kód se do KBCODE nastaví vždy na začátku programové smyčky a na jejím konci se automaticky vynuluje. Proměnná KBCODE je tedy po stisku klávesy aktivní právě po dobu jednoho průchodu programovou smyčkou. K opětovnému nastavení KBCODE je třeba tlačítko pustit a znovu zmáčknout (výjimkou je autorepeat - viz dále). Pokud je klávesnice v klidu, je KBCODE=0.

Kódování kláves

klávesa	KBCODE	ASCII	klávesa	KBCODE	ASCII	klávesa	KBCODE	ASCII
←	1		F1	8		4	52	"4"
→	2		F2	9		5	53	"5"
↑	5		F3	10		6	54	"6"
↓	6		0	48	"0"	7	55	"7"
ESC	3		1	49	"1"	8	56	"8"
ENT	4		2	50	"2"	9	57	"9"
+ / -	7		3	51	"3"	.	46	"."

5.1.1 Autorepeat

Aby nebylo nutné pro určité akce (např. pro posuv kursoru, rolování menu apod.) vícekrát mačkat tutéž klávesu, je možné nastavit "AUTOREPEAT". Pokud stiskneme některé tlačítko a držíme je stisknuté, potom po určité (nastavitelné) prodlevě bude do KBCODE znovu nastavován kód tlačítka (s nastavitelnou rychlostí opakování). Efekt je tentýž jako bychom neustále rychle mačkali tlačítko.

Funkce se zapíná nastavením bitu KBREPEN, počáteční prodleva se nastavuje registrem KBDELAY (v desítkách milisekund) a délka intervalu mezi jednotlivými "pseudo-stisky" registrem KBREPEAT (v desítkách milisekund).

```
  Příklad zapnutí funkce autorepeat s počáteční prodlevou 1
sec. a opakováním po 0.2 sec. :
  KBREPEN; KBDELAY = 100; KBREPEAT = 20;
```

5.1.2 Akustická indikace

Stisk tlačítka (a případně i "pseudo-stisky" při zapnuté funkci AUTOREPEAT) je možné automaticky indikovat akusticky krátkým pípnutím vestavěného bzučáku. Funkce se zapíná nastavením funkčního bitu KBSOUND (bit je automaticky nastaven vždy po zapnutí).

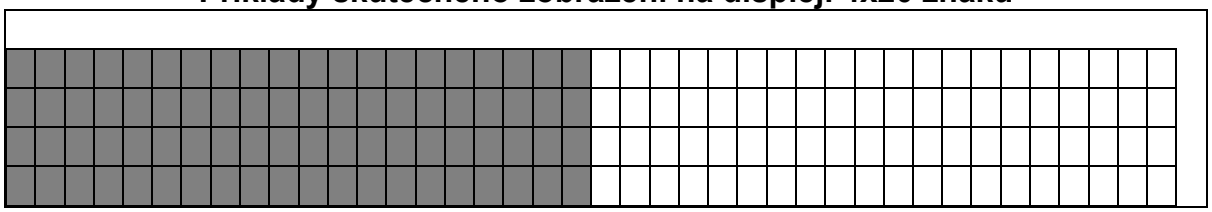
5.2 Displej

Texty, čísla nebo uživatelské symboly se vždy tisknou na virtuální obrazovku velikosti 4 řádky x 40 znaků. Pozice znaků na obrazovce se počítají od levého horního rohu počínaje nulou.

Číslování pozic znaků na virtuální obrazovce

0																																													39
40																																													79
80																																													119
120																																													159

Příklady skutečného zobrazení na displeji 4x20 znaků



Pro tisk na displej je vyhrazena speciální proměnná DISPLAY, do které lze kromě konstant a proměnných přiřazovat i text (píše se do uvozovek). Pozici, odkud se bude vypisovat, lze volně nastavit pomocí funkční proměnné POSITION. Pokud chceme tisknout položky za sebou, není třeba počítat nové hodnoty do proměnné POSITION, neboť obsah této proměnné se automaticky po tisku každého znaku zvětší o 1.

Tisk textů je přímý, tj. to co je napsáno v uvozovkách a přiřazeno do proměnné DISPLAY je zobrazeno od pozice dané proměnnou POSITION.

5.3 Formáty zobrazování číselných hodnot

Jazyk SIMPLE3 podporuje práci s proměnnými typu bit, word, integer a real. Při výstupu jakéhokoliv čísla na displej je třeba v uživatelském programu zvolit formát jakým je danné číslo zobrazováno. Volba formátu se provádí naplněním proměnné FORMAT příslušnou konstantou. Jednotlivé řády formátu určují specifické vlasnosti zobrazení.

5.3.1 Typ BIT

Pro tento číselný typ mají smysl pouze formáty 0 až 4. Zobrazení čísla typu bit v závislosti na obsahu proměnné FORMAT je patrné z následující tabulky.

Formát	Způsob zobrazení
0	zobrazí hodnotu 0 nebo 1
1	zobrazí znak “L” nebo “H”
2	zobrazí znak - nebo +
3	zobrazí nápis LOW nebo HIGH
4	zobrazí nápis VYP nebo ZAP

5.3.2 Typ WORD

Řád jednotek se při zobrazování tohoto typu neuplatní. Řád desítek formátu udává nejmenší počet míst, na nejž je číslo zobrazováno. Pojem “nejmenší počet míst” znamená, že pokud má zobrazované číslo větší počet platných číslic než udává formát, jsou samozřejmě zobrazovány všechny číslice. Řád stovek určuje způsob zarovnání. Je-li nulový, číslo je zarovnáno v pravo, je-li jedničkový, provádí se zarovnání vlevo. Řád stovek hodnoty větší než 1 nemá pro čísla typu word smysl. Řád tisíců různý od nuly nemá u formátu čísel typu word význam.

Příklad:

```
  D0 = 123;
  FORMAT = 50;
  DISPLAY = D0; //Cislo se zobrazi ve tvaru __123.(Tedy na pet
mist, dve mezery na zacatku)
  FORMAT = 140;
  DISPLAY = D0; //Cislo se zobrazi ve tvaru 123_.(Ted na ctyri
mista, jedna mezera na konci)
  FORMAT = 10;
  DISPLAY = D0; //Cislo se zobrazi ve tvaru 123.(Cislo ma vice
platnych cilic nez urcuje format)
```

5.3.3 Typ INTEGER

Je-li řád tisíců jedna, zobrazuje se u kladného čísla na začátku znaménko + (U záporných čísel se znaménko - zobrazuje vždy). Řád desítek formátu udává nejmenší počet míst, na nejž je číslo zobrazováno. Tento počet míst je uvažován včetně znaménka. Řád stovek určuje způsob zarovnání. Je-li nulový, číslo je zarovnáno vpravo, je-li jedničkový, provádí se zarovnání vlevo. Řád stovek hodnoty větší než 1 nemá pro čísla typu integer smysl.

Příklad:

```
  Cislo = 456;
  FORMAT = 40;
  DISPLAY = Cislo;   //Cislo se zobrazi ve tvaru _456.(Tedy na
ctyri mista, jedna mezera na zacatku, bez znamenka)
  FORMAT = 1150;
  DISPLAY = Cislo;   //Cislo se zobrazi ve tvaru +456_.(Ted na
pet mist, jedna mezera na konci, se znamenkem)
```

5.3.4 Typ REAL

Je-li řád tisíců jedna, zobrazuje se u kladného čísla na začátku znaménko + (U záporných čísel se znaménko - zobrazuje vždy). Řád jednotek určuje počet zobrazovaných desetinných míst. Řád desítek formátu udává nejmenší počet míst, na nejž je číslo zobrazováno. Tento počet míst je uvažován včetně znaménka a řádové tečky. Řád stovek určuje způsob zarovnání. Je-li nulový, číslo je zarovnáno vpravo, je-li jedničkový, provádí se zarovnání vlevo. Je-li řád stovek roven 2, potom je číslo zobrazováno v semilogaritmickém tvaru.

Příklad:

```
  Cislo = 78.9;
  FORMAT = 62;
  DISPLAY = Cislo;   //Cislo se zobrazi ve tvaru _78.90.(Tedy
na sest mist, jedna mezera na zacatku, dve destina mista)
  FORMAT = 1072;
  DISPLAY = Cislo;   //Cislo se zobrazi ve tvaru _+78.90.(Tedy
na sedum mist, jedna mezera na zacatku, znamenko polarity, dve
desetna cisla)
  FORMAT = 161;
  DISPLAY = Cislo;   //Cislo se zobrazi ve tvaru 78.9__.(Tedy
na sest mist, dve mezery na konci, jedno desetinne misto, bez
znamenka)
  FORMAT = 203;
  DISPLAY = Cislo;      //Cislo    se    zobrazi    ve    tvaru
7.890E01.(Tedy v semilogaritmickem tvaru, tri desetina mista)
```

5.3.5 Formát pro tisk znaků

Je-li použit formát 220, je vstupní hodnota interpretována jako číslo znaku a vytiskne na display jeden znak daný tímto číslem. Tento formát umožňuje tisknout i znaky, které nejsou běžně dostupné a nelze je tedy při psaní programu napsat z klávesnice do zadání textu pro textový výstup. Kód znaku musí být v rozsahu 0 ... 255.

Kód	Typy znaků
0...7	vlastní předefinovatelné znaky, grafické symboly
32...122	abecední a číslicové znaky, kódované podle mezinárodního standardu ASCII
160...255	další užitečné znaky (např. japonská abeceda Katakana)

Například znak “C” lze v jazyce SIMPLE3 zobrazit několika způsoby:

```
DISPLAY = "C";
FORMAT=220; DISPLAY=67;
D1=67; FORMAT=220; DISPLAY=D1;
Tabulka kódů nejpoužívanějších znaků  pro LC display:
```

kód	znak	kód	znak	kód	znak	kód	znak	kód	znak	kód	znak	kód	znak
32		48	0	64	@	80	P	96	`	112	p	161	o
33	!	49	1	65	A	81	Q	97	a	113	q	219	ˆ
34	"	50	2	66	B	82	R	98	b	114	r	223	o
35	#	51	3	67	C	83	S	99	c	115	s	224	a
36	$	52	4	68	D	84	T	100	d	116	t	225	ä
37	%	53	5	69	E	85	U	101	e	117	u	226	b
38	&	54	6	70	F	86	V	102	f	118	v	227	e
39	'	55	7	71	G	87	W	103	g	119	w	228	m
40	(	56	8	72	H	88	X	104	h	120	x	229	s
41	)	57	9	73	I	89	Y	105	i	121	y	230	r
42	*	58	:	74	J	90	Z	106	j	122	z	235	x
43	+	59	;	75	K	91	[	107	k	123	{	239	ö
44	,	60	<	76	L	92		108	l	124	|	244	W
45	-	61	=	77	M	93	]	109	m	125	}	245	ü
46	.	62	>	78	N	94	^	110	n	126	®	246	p
47	/	63	?	79	O	95	_	111	o	127	¬	255	ˆ

5.3.6 Definování vlastních grafických symbolů na LC disleji

Terminály řady “T” a kompaktní řídící stanice “PP” s LC displejem umožňují vytvořit až 8 vlastních grafických symbolů, které se dají vytisknout na displeji jako kterýkoliv jiný znak. Znaky jsou v tabulce znaků kódovány pod čísly 0 až 7 a lze je tisknout prostřednictvím formátu 221. Tuto zajímavou funkci můžeme využít např. pro tvorbu a tisk znaků české abecedy.

Všechny znaky jsou tvořeny maticí z 8 grafických řádků, kde každý řádek obsahuje 5 bodů. U stanic s LED displejem je uživatelský symbol tvořen pouze jedním číslem, které odpovídá binárnímu kódu pro sedmisegmentový znak. U standardní pevné sady znaků se spodní řádek nevyužívá. U sady programovatelných znaků můžeme využít všech 8 řádek. Celkem můžeme definovat 64 grafických řádků, tvořících 8 kompletních znaků.

Definování znaků začíná nastavením hodnoty 221 do proměnné FORMAT. Po nastavení formátu 221 se displej přepne z režimu tisku do režimu programování znaků. Proměnné DISPLAY a POSITION mají v tomto režimu jinou funkci. Proměnná POSITION určuje, který grafický řádek budeme měnit a do proměnné DISPLAY zapisujeme data, určující obsah řádku. POSITION i v tomto režimu funguje tak, že po zápisu grafického řádku se automaticky zvětší o 1 a posune se tak na další řádek. Programování znaků ukončíme nastavením proměnné FORMAT na jinou hodnotu. Po návratu z programovacího režimu se do proměnné POSITION nevrací původní hodnota, je tedy nutné ji před dalším tiskem opět nastavit.

Umístění pozic grafických řádků

znak 0	znak 1	znak 2	znak 3	znak 4	znak 5	znak 6	znak 7
0	8	16	24	32	40	48	56
1	9	17	25	33	41	49	57
2	10	18	26	34	42	50	58
3	11	19	27	35	43	51	59
4	12	20	28	36	44	52	60
5	13	21	29	37	45	53	61
6	14	22	30	38	46	54	62
7	15	23	31	39	47	55	63

Každý bod řádku má svoji numerickou váhu (odpovídá binárnímu kódování). Numerickou reprezentaci jednoho grafického řádku získáme sečtením hodnot odpovídajících jednotlivým bodům.

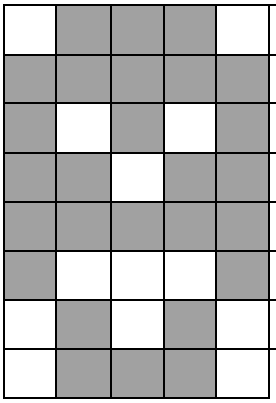
číselné vyjádření bodů na grafickém řádku je toto:

16	8	4	2	1

Například tento grafický řádek  má číselné vyjádření 25.

Příklad:

Chceme zadefinovat grafický symbol (např. podle následujícího obrázku) a vytisknout na LC displej. Znak chceme zadefinovat do tabulky znaků například pod kódem 2.

GRAFICKÁ PODOBA	DATA
	14 31 21 27 31 17 10 14

```
FORMAT=221;// formát pro definování znaků
POSITION=16;    // 0. grafický řádek znaku 2
DISPLAY=14;// zápis grafických řádků
DISPLAY=31;
DISPLAY=21;
DISPLAY=27;
DISPLAY=31;
DISPLAY=17;
DISPLAY=10;
DISPLAY=14;
FORMAT=220;// budeme tisknout znak podle kódu
POSITION=0;// na 1. pozici na 1. řádku
DISPLAY=2; // bude se tisknout znak číslo 2
```

5.4 Princip zobrazování

Tisk na virtuální obrazovku a vlastní fyzický zápis dat do obvodů displaye jsou dva zcela nezávislé děje, které běží v automatu paralelně. Odezva displaye je totiž pomalá a kdyby měl uživatelský program čekat, až se skutečně zobrazí nějaký delší text, došlo by v daném bodě k jeho neúnosnému zdržení. Celý tisk přes proměnnou DISPLAY se proto uloží do paměti RAM (zde je ta virtuální obrazovka) a zároveň s programem běží na pozadí proces, který tyto znaky zobrazuje.

Tato metoda sice nezdržuje uživatelský program, avšak skýtá jednu záludnost. Chceme-li například smazat řádek displaye a vytisknout na něj novou informaci, můžeme to provést např. takto :

```
POSITION=40;              // budeme pracovat s 2. řádkem

DISPLAY="               "; //tisk mezer přes celý řádek

POSITION=40;

DISPLAY="NOVY TEXT";      //na začátek řádku nový text
```

Přepsání celého displeje trvá asi 200 ms. Pokud budeme tuto proceduru volat často (alespoň 3x až 4x za sekundu), může se stát, že zrovna po vytisknutí prázdného řádku se několik těchto znaků přenese na displej a bude trvat asi 200 ms, než dojde k jejich přepisu novou hodnotou. A shodou náhod může být v tento okamžik obslužný program zase v bodě, kdy do inkriminované oblasti opět píše mezery... Obraz potom různě chaoticky problikává.

Řešením je buď omezit frekvenci volání takové procedury (max. 2x za sekundu), nebo tisknout na display tak, aby nedocházelo bezprostředně po sobě k přepisu týchž pozic různou informací. Např. takto :

```
POSITION=40;              //budeme pracovat s 2. řádkem

DISPLAY="NOVY TEXT      "; //tisk textu a mezer až do konce
```

6 Síťová vrstva

Všechny řídící prostředky vyráběné firmou HYPEL je možno bez úprav či modifikací přímo použít k vytváření decentralizovaných řídících systémů s distribuovanou inteligencí.

Síťová vrstva umožňuje komunikaci mezi automaty navzájem, přičemž si jsou všechny stanice v síti rovnocenné. Stanic může být současně zapojeno ve společné síti max. 31. Přenášení dat mezi stanicemi je realizováno dvěma základními mechanismy: Sdílením síťových proměnných a posíláním přímých adresných zpráv.

6.1 Síťové proměnné

Aby se dalo jednoduše komunikovat mezi automaty pomocí prostředků jazyka SIMPLE3, byl zvolen tento jednoduchý způsob :

Horní polovina uživatelských bitů M (tedy M63 až M127) a horní polovina uživatelských proměnných D (tedy D32 až D63) je sdílená všemi automaty v síti. Pokud tedy program v automatu zapíše hodnotu do některé této proměnné (nastaví nebo nuluje bit - např. M64 ; M65' nebo zapíše hodnotu do proměnné např. D32=3568; D33=D11 / 120 apod.), automat při nejbližší možné příležitosti odvysílá do sítě adresu a hodnotu této proměnné. Ostatní automaty tuto zprávu zachytí a automaticky si do těchto proměnných zapíší nové hodnoty.

Tato metoda je velmi jednoduchá a lehce pochopitelná, má však svá úskalí. Jednak je komunikace poměrně pomalá a přenos dat může tímto způsobem trvat za nepříznivých okolností i jednu sekundu. Dále není zaručena přesná sekvence přenosu síťových proměnných tak, jak by to odpovídalo pořadí zápisů do proměnných z programu automatu. Nejpodstatnější je ovšem fakt, že jelikož jsou obsahy síťových proměnných posílány všem stanicím v síti současně, nemohou být přijetí těchto zpráv již od principu potvrzováno. Jiným slovy máme sice jistotu, že nemohou dorazit na místo určení chybná data, ale nemůžeme zaručit, že aktuálně platná data vůbec dorazí. Přenos proměnné je aktivován vždy pouze po jejím zapsání z programu v automatu. Pokud však právě v tuto chvíli dojde ke krátkodobému výpadku sítě, ostatní automaty v síti se o změně obsahu dané síťové proměnné již nedozvědí. Proto lze doporučit tuto jednoduchou metodu bez obav pouze v tom případě, že je aktuální hodnota do síťové proměnné zapisována neustále.

Například pokud máme decentralizovaný řídící systém v němž jedna stanice měří na jednom svém analogovém vstupu teplotu a chceme, aby byla tato teplota “viditelná” pro všechny automaty v síti provedeme to takto: V hlavním těle programové smyčky zapíšeme například následující řádek.

```
D32=I0;
```

Potom budeme moci ve všech automatech v síti číst z proměnné D32 aktuální hodnotu měřené teploty. Pokud v tomto případě dojde ke krátkodobému výpadku komunikace nebude to na závadu, protože hodnota proměnné bude aktualizována bezprostředně po jejím obnovení. Použití metody sdílených proměnných je v tomto případě na místě.

6.2 Adresné zprávy

Přenos dat mezi automaty v síti pomocí adresných zpráv je z hlediska uživatelského programu poněkud komplikovanější, ale má mnoho předností. Především můžeme zapisovat z kteréhokoliv automatu v síti do kterékoliv proměnné libovolného jiného automatu a obdobně i číst. Nejsme tedy omezeni jen na oblast síťových proměnných a můžeme provést zápis například i do proměnných výstupních či například deklarovaných proměnných (var). Tato volnost sebou samozřejmě přináší větší rizika při neopatrném použití. Druhou značnou výhodou této metody je fakt, že přenášené zprávy jsou vždy potvrzované a tudíž máme vždy jistotu, že byla požadovaná data přenesena do určené stanice. Velmi podstatné také je, že přenos dat je podstatně rychlejší oproti metodě sdílených proměnných.

K vysílání adresných zpráv slouží sada funkčních proměnných . Jsou to:

SENDCODE, SENDADDR, MEMADDR, BITADDR, SENDDATA, SENDBIT, MYADDR

6.1.1 SendAddr

Do této proměnné zapisujeme adresu stanice již je zpráva určena. To znamená, že je možno zapsat hodnotu 0 až 30. Je nevhodné zapisovat do proměnné vlastní adresu.

6.1.1 MemAddr

Obsah této proměnné určuje adresu v paměti vybrané stanice na kterou se bude zapisovat případně z ní číst.

6.1.1 BitAddr

Obsah této proměnné se uplatní pouze u operací s bitovými proměnnými. Určuje adresu bitu v bytu určeném proměnnou MemAddr. Může nabývat hodnot 0 až 7. Jelikož se proměnné adresují přímo v RAM automatu je zde uvedena tabulka umístění základních proměnných. Adresy jsou udány jak v hexadecimální tak dekadické reprezentaci.

Proměnné	Bytová adresa MemAddr	Bitová adresa BitAddr
X0 až X7	200H = 512d	0 až 7
X8 až X15	201H = 513d	0 až 7
X16 až X23	202H = 514d	0 až 7
X24 až X31	203H = 515d	0 až 7
Y0 až Y7	204H = 516d	0 až 7
Y8 až Y15	205H = 517d	0 až 7
Y16 až Y23	206H = 518d	0 až 7
Y24 až Y31	207H = 519d	0 až 7
I0 až I31	00H až 3EH = 0d až 62d	neuplatní se
O0 až O31	40H až 7EH = 64d až 123d	neuplatní se
M0 až M7	208H = 520d	0 až 7
M8 až X15	209H = 521d	0 až 7
M16 až M23	20AH = 522d	0 až 7
M24 až M31	20BH = 523d	0 až 7
D0 až D31	80H až 0BEH = 128d až 190d	neuplatní se
Deklarované proměnné	od 1000H = 4096d	0 až 7

Proměnné volně deklarované (var) jsou v RAM automatu ukládány od adresy 4096. Při jejich využití v síťovém provozu je ne vhodné postupovat takto. Nejdříve napíšeme zdrojový text cílové stanice. Deklarace proměnných, které mají být využity k síťovému přenosu deklarujeme jako první. Zdrojový text bude i nadále možno při ladění aplikace měnit, ale jakékoliv nové proměnné je třeba deklarovat až definice původní. Provedeme překlad programu a adresy daných proměnných nalezneme přímo ve vygenerovaném "dnl" souboru.

6.1.1 SendData

Při vysílání je přenášena do cílové stanice obsah právě této proměnné. Naopak při čtení ze sítě jsou do této proměnné ukládána přijatá data. Používáme-li operace pro přenos bytů je využit pouze spodní byte této proměnné.

6.1.1 SendBit

Toto je funkční obdoba proměnné SendData, ale pro bitové operace. Je-li SendBit nastaven je nastaven i cílový bit v druhém automatu a nopak. Při operacích čtení je SendBit naplněn hodnotou příslušného bitu druhého automatu.

6.1.1 SendCode

Obsah této proměnné určuje jaká síťová operace se bude provádět. Zápisem příslušného kódu do této proměnné se zahájí provádění této operace a je-li úspěšně provedena je obsah proměnné automaticky nulován.

Typ Operace	SendCode
Zápis bitu	1
Čtení bitu	2
Zápis bytu	3
Čtení bytu	4
Zápis wordu	5
Čtení wordu	6
Úspěšný přenos	0

6.2.1 Odesílání zpráv

Pokud chceme z programu zapsat do proměnné jiného automatu nejdříve zkontrolujeme je-li obsah proměnné SendCode roven nule, což znamená, že již byla úspěšně odeslána předešlá zpráva. Odesílání předešlé zprávy můžeme též stornovat tím, že do SendCode zapíšeme nulu. Potom nastavíme adresu cílové stanice do proměnné SendAddr, adresu cílové proměnné do MemAddr, případně i do BitAddr, (jedná-li se o zápis do bitové proměnné) a požadovanými daty naplníme proměnnou SendData, případně SendBit. Vlastní odesílání se zahájí zápisem příslušného kódu do proměnné SendCode. Obsah SendCode se automaticky vynuluje po úspěšném přenosu zprávy. Například chceme-li zapsat z programu nastavit automatu s adresou 5 na analogovém výstupu O0 hodnotu 1500 provedeme to takto:

```
symbol
  5 # WriteWord;
end
if SendCode=0 then
  SendAddr=5; MemAddr=64; SendData=1500;
  SendCode=WriteWord;
endif
```

Při čtení z proměnné druhého automatu Naplníme proměnné SendAddr, MemAddr, případně BitAddr. Dále nastavíme příslušný kód operace do proměnné SendCode. Dále testujeme obsah proměnné SendCode. Jakmile je vynulována čteme z proměnné SendData nebo SendBit platná data. Chceme-li například číst hodnotu vstupu X1 z automatu s adresou 7 a tuto hodnotu stále ukládat do proměnné M0 provedeme to takto:

```
symbol
  2 # ReadBit;
end
if SendCode=0 then
  M0=SedBit;
  SendAddr=7; MemAddr=512; BitAddr=1;
  SendCode=ReadBit;
endif
```

6.3 Příprava automatů pro provoz v síti

Nejprve je třeba všem automatům nastavit stejnou komunikační rychlost a různé adresy. Nastavení se zapisuje do EEPROM v automatech a provádí se podpůrným programem SETPES3 - viz kap. "Podpůrné programy".

Tento program vyžaduje připojení jen jedné stanice na lince. To lze provést např. tak, že postupně připojujeme jeden automat po druhém a provádíme nastavení, anebo zapojíme kompletně celou síť včetně instalace všech automatů a nastavování provádíme s postupným zapínáním a vypínáním jednotlivých automatů.

6.3.1 Volba komunikační rychlosti

Musí být u všech automatů stejná a lze si vybrat z hodnot 2400 Bd, 9600 Bd, 19200 Bd a 57600 Bd. Nastavování se provádí programem SETPES. Pokud celková délka kabeláže nepřesahuje 100 metrů, lze provozovat síť na rychlosti 57600 Bd, při délkách do 500 metrů doporučujeme použití rychlosti 19200 Bd, při silném rušení nebo větších délkách kabeláže je vhodné použít 9600 Bd nebo 2400 Bd.

Samozřejmě na komunikační rychlosti přímo závisí datová propustnost sítě a s tím související např. odezvy automatů na změny sdílených síťových proměnných.

7 Tipy pro užívání SIMPLE3

Některé konstrukce, které oproti jazyku SIMPLE2 přináší SIMPLE3 mohou v některých případech značně zjednodušit zápis zdrojového textu a tím zefektivnit práci na tvorbě aplikace.

7.1 Využívání definice konstant ENUM

Definice konstant ENUM umožňuje jednoduše nadefinovat množinu konstant, které mají samovysvětlující identifikátory, zvyšující přehlednost programu.

Příklad

```
CONST
 ENUMNic,Pondeli,Utery,Streda,Ctvrtek,Patek,Sobota,Nedele ;
END

CASE Den
 OF  Pondeli   THEN  DISPLAY="Po"
 OF  Utery     THEN  DISPLAY="Ut"
 OF  Streda    THEN  DISPLAY="St"
 OF  Ctvrtek   THEN  DISPLAY="Ct"
 OF  Patek     THEN  DISPLAY="Pa"
 OF  Sobota    THEN  DISPLAY="So"
 OF  Nedele    THEN  DISPLAY="Ne"
ELSE                 DISPLAY="??"
ENDCASE
```

Pojmenování konstant může posloužit pro udržení přehledu v polích:

Příklad

```
  CONST
   ENUM   Chodby,Telocvicna,Kuchyn,Ucebny; //regulacni okruhy
END

VAR
     PozadHodnota : array [Ucebny] of real ;
END
PozadHodnota[Telocvicna] = 18.0;
```

V případě, že při úpravách programu dojde k vypuštění okruhu, nebo naopak k rozšíření počtu okruhů, jsou úpravy mnohem jednodušší.

7.2 Konečný automat

Každý řídicí program je, ať již si to uvědomujeme či ne, realizací tzv. konečného automatu. Ten má soubor vstupů, výstupů a vnitřních stavů. V každém „taktu“ dochází na základě hodnot vstupů a vnitřního stavu ke generování hodnot na výstupech a k určení nového vnitřního stavu. Vstupy a výstupy bývají jasně definovány již samotným zadáním řídicí úlohy. Volnost je dána programátorovi při definování počtu a způsobu reprezentování vnitřních stavů a tím i způsobu popisu funkcí, které rozhodují o výstupu a novém vnitřním stavu programu.

Častým způsobem definování vnitřních stavů je soubor „flagů“, tedy bitových proměnných, které se různě nastavují a nulují, jak je potřeba. Funkce přechodu mezi stavy jsou pak soustavou logických rovnic. Tento způsob je obvykle kombinován s hodnotami číselných proměnných, které se pak porovnávají s různými konstantami.

SIMPLE3 nabízí možnost rozlišovat stavy hodnotami jedné nebo více stavových proměnných typu WORD nebo INT. Jednotlivé stavy vyjmenovat v definici konstant pomocí ENUM, aniž bychom se starali o konkrétní hodnoty, které jim jsou přiřazeny. Struktura CASE pak větví program podle hodnoty stavové proměnné.

Tento styl doporučujeme především pro zvýšení přehlednosti a čitelnosti programu.

Mějme jednoduchou úlohu jako součást nějakého složitějšího systému - poruchová signalizace s potvrzením. Požadujeme následující funkci.

- ✓ Pokud není porucha, signálka nesvítí
- ✓ Objeví-li se porucha, signálka začne blikat.
- ✓ Pomine-li porucha sama, signálka zůstane rozsvícená až do stisku potvrzení.
- ✓ Pokud poruchu kvitujeme, signálka svítí až do pominutí poruchy.

Zmíněnou metodou můžeme úlohu řešit například takto.

```
  CONST
     ENUM  BezPoruchy,JePorucha,Pominula,Potvrzena ;
     50    #    PeriodaBlikani ; // v desitkach milisekund
END

VAR  Stav           : WORD ;
     BlikajiciBit   : BIT ;
     X0    #    Kvitace ;
     X1    #    Porucha ;
     Y0    #    Signalka ;
END

{-- programova smycka ---------------------------------------}
```

```
IF RESET  THEN   TEN0; TPA0'; TOE0; T0=0; RESET'; ENDIF
IF T0>=PeriodaBlikani  THEN !BlikajiciBit; T0=0;  ENDIF

CASE Stav
     OF   BezPoruchyTHEN Signalka';
                  IF Porucha  THEN Stav=JePorucha;  ENDIF
     OF   JePorucha THEN Signalka = BlikajiciBit ;
                  IF Porucha' THEN Stav=Pominula;   ENDIF
                  IF Kvitace  THEN Stav=Potvrzena;  ENDIF
     OF   Pominula  THEN Signalka;
                  IF Kvitace  THEN Stav=BezPoruchy; ENDIF
                  IF Porucha  THEN Stav=JePorucha;  ENDIF
     OF   Potvrzena THEN Signalka;
                  IF Porucha' THEN Stav=BezPoruchy; ENDIF
ENDCASE

END
```

V uvedeném příkladě jsme zanedbali zpoždění jednoho cyklu programu mezi zjištěním poruchy a reakcí signálky. Pokud by zpoždění z nějakých důvodů vadilo, musela by být reakce signálky uvedena již v podmíněném příkazu IF. Chtěli jsme být co nejstručnější.

8 Podpůrné programy

Následující programy přímo komunikují s automatem, je tedy nutné propojit automat přes převodník RS232<->RS485 k sériovému portu PC. Lze použít např. typ PES-CA1, standardně dodávaný k automatům HYPEL.

8.1 SETPES3

Tento program je určen k nastavování důležitých systémových parametrů v EEPROM automatu PES. Pro jeho funkci je nutné, aby na lince byl zapojen právě jeden automat. Program je důležitý zejména pro přípravu automatů k zapojení do sítě.

Program se spouští z příkazové řádky takto:

```
SETPES /COMx /ADDRx /BAUDx /TIME /LOCK
```

/COMx	udává číslo portu, na kterém je připojen automat (může být /COM1 až /COM4) - musí být vždy uveden !!
/ADDRx	přiřazuje automatu síťovou adresu - povolené hodnoty 0..30
/BAUDx	nastavuje baudovou rychlost (musí být u všech automatů shodná) povolené hodnoty 2400, 9600, 19200 a 57600
/TIME	přenese aktuální hodnotu reálného času z PC do automatu
/LOCK	provede uzamčení paměti programu, dále již nelze zavádět nový program, či měnit systémové parametry.
/UNLOCK	odemčení paměti programu pro zápis, dále již lze zavádět nový program, či měnit systémové parametry.

Příklad:

```
SETPES /COM2 /ADDR15 /BAUD19200 /TIME
```

Jediný povinný parametr je /COMx, ostatní parametry lze použít dle potřeby. Postup při volbě parametrů a při přípravě automatů ke spojování do sítě - viz výše - kap. "Funkce síťové vrstvy".

Pokud nevyužíváte automat v síti spolu s dalšími automaty, není používání programu SETPES nutné.

8.2 LOADER3

Program LOADER3.EXE, dodávaný spolu s překladačem jazyka SIMPLE3, provádí tzv. download (zatažení) přeloženého programu z počítače PC do automatu.

Loader se spouští z příkazové řádky takto:

```
LOADER2 /COMx SOUBOR
```

SOUBOR	název přeloženého programu, který je třeba spustit (musí to být soubor typu *.DNL) - název je možno uvést i bez přípony - příponu .DNL si program doplní sám
/COMx	udává číslo portu, na kterém je připojen automat (může být /COM1 až /COM4) - musí být vždy uveden !!
/ADDRx	nastavuje adresu automatu - do automatu s touto adresou se bude provádět download
/TIME	přenese aktuální hodnotu reálného času z PC do automatu

Po spuštění programu nejprve probíhá detekce přítomnosti stanic na lince a testování přenosové rychlosti, na které se momentálně v síti komunikuje. Program se sám přizpůsobí pro nalezenou rychlost.

Pokud je na linku připojen jen jeden automat, loader si sám najde jeho adresu a na tuto adresu provede download. Pokud je v síti zapojeno více automatů, je již nutné adresu specifikovat. To lze dvěma způsoby:

- ✓ Pokud byla ve zdrojovém textu zapsána direktiva NetAddr s příslušnou adresou, je tato hodnota automaticky přenesena do *.DNL souboru a loader bez dalších testů linky použije tuto adresu pro download.

- ✓ Nastavení adresy parametrem /ADDR přímo z příkazové řádky - loader potom provádí download na tuto adresu (a to i v případě, že ve zdrojovém textu byla již nastavena nějaká adresa direktivou NetAddr !)

Po přenosu programu do automatu LOADER3 sám zajistí spuštění programu v automatu a nastavení příznaku AUTORUN (tento příznak zajistí automatický start programu po každém zapnutí automatu).

Příklad spuštění programu VYTAH (automat s adresou 12 připojen na COM2):

```
LOADER3 VYTAH /COM2 /ADDR12
```

Poznámka: Pokud po spuštění LOADER3 hlásí, že linka nekomunikuje, přesvědčte se, zda je automat zapnutý a zda souhlasí číslo COM portu.

8.3 VIEWER3

Tento program slouží k zobrazování a editaci proměnných za chodu programu v automatu. Je užitečný zejména tehdy, když se spuštěný program nechová podle představ svého tvůrce a je třeba zjistit, co se děje v proměnných používaných v programu. Parametry pro spuštění programu VIEWER3.EXE jsou podobné jako pro LOADER3. Jako vstupní soubor pro VIEWER3.EXE slouží tentýž *.DNL soubor jako pro loader (v *.DNL souborech jsou totiž kromě kódu pro interpreter ještě údaje o použitých symbolických proměnných).

Viewer se spouští z příkazové řádky takto:

```
VIEWER2 SOUBOR /COMx
```

SOUBOR	název přeloženého programu, který je třeba monitorovat (musí to být soubor typu *.DNL) - název je možno uvést i bez přípony - příponu .DNL si program doplní sám
/COMx	udává číslo portu, na kterém je připojen automat (může být /COM1 až /COM4) - musí být vždy uveden !!
/ADDRx	nastavuje adresu automatu - do automatu s touto adresou se bude provádět download
/MORE	zapíná zobrazování nepoužitých proměnných, jako jsou některé spec. funkční proměnné, všechny vstupy/výstupy apod.
/FAST	rychlejší občerstvování zobrazených informací (viz níže)

Po spuštění programu nejprve probíhá detekce přítomnosti automatů na lince a testování přenosové rychlosti, na které se momentálně v síti komunikuje. Program se sám přizpůsobí pro nalezenou rychlost.

Pokud je na linku připojen jen jeden automat, viewer si sám najde jeho adresu. Pokud je v síti zapojeno více automatů, je již nutné adresu specifikovat. To lze dvěma způsoby:

- ✓ Pokud byla ve zdrojovém textu zapsána direktiva NetAddr s příslušnou adresou, je tato hodnota automaticky přenesena do *.DNL souboru a viewer bez dalších testů linky tuto adresu použije.

- ✓ Nastavení adresy parametrem /ADDR přímo z příkazové řádky - viewer potom použije tuto adresu (a to i v případě, že ve zdrojovém textu byla již nastavena nějaká adresa direktivou NetAddr !)

Po spuštění nabízí VIEWER3 vlevo stručné menu funkcí a vpravo výpis symbolických názvů proměnných používaných v programu. V seznamu se lze pohybovat šipkami nahoru a dolů, klávesou ENTER lze vyvolat okno pro modifikaci proměnné (program přitom kontroluje přípustný rozsah hodnot, nelze tedy např. do binární proměnné přiřadit číslo 2). Dále viewer umožňuje pozastavení programu (PAUSE ON/OFF) a restartování programu.

8.3.1 Seznam proměnných

Do seznamu v pravém okně jsou zahrnuty všechny proměnné, které jsou v programu použity.

Pokud program využívá některý časovač, budou v seznamu i všechny funkční bity, které se ho týkají. Pokud program využívá reálný čas, objeví se v seznamu zároveň všechny registry reálného času.

V seznamu se dále ještě objevují některé další systémové registry.

8.3.2 Frekvence občerstvování hodnot proměnných

Vzhledem k tomu, že COM-port počítače PC není schopen zvládnout provoz protokolem PESNET 3.32 na vyšších rychlostech, je zobrazování proměnných realizováno dávkovým způsobem:

- ✓ Přeruší se síťová komunikace mezi automaty navzájem a nastaví se mód "single-master", kde master je v tomto případě PC.
- ✓ Vyčtou se naráz hodnoty všech proměnných které jsou momentálně na obrazovce.
- ✓ Opět se nastaví mód "multi-master" a rozeběhne se komunikace mezi automaty.

Tento způsob zobrazování značně snižuje průchodnost sítě a je tedy třeba počítat s výrazně delší odezvou síťové komunikace, pokud je v síti spuštěn program VIEWER2.

Standardní interval občerstvování zobrazovaných hodnot je při rychlosti 57600 Bd asi 2-3 sec, při nižších komunikačních rychlostech je tato doba příslušně delší. Tento interval lze zkrátit použitím parametru /FAST z příkazové řádky - pak se interval zkracuje zhruba na 1/4.

Poznámka: Pokud po spuštění VIEWER2 hlásí, že linka nekomunikuje, příčina je patrně buď ve špatném propojení linky, nebo nesouhlasí číslo COM portu.

8.4 Konvert

Pro případ, že chcete přeložit program, původně určený pro starší verzi SIMPLE2, je připraven program konvert.exe. Provede všechny potřebné úpravy tak, aby byl program akceptovatelný překladačem SIMPLE 3. Dbejte, aby byl v původním programu na konci příkaz END. Program se volá:

```
konvert.exe   původní_soubor   cílový_soubor
```

Shrnutí úprav, které konvert provede na zdrojovém programu pro předchozí verzi:

- ✓ Doplní ENDIFy na konec podmíněných příkazů
- ✓ Mění znak řádkového komentáře ze středníku na dvojznak //
- ✓ Změní příkazové oddělovače. Místo znaků ´:´ se používá středník ´;´.
- ✓ Uzavře deklarace do bloků vymezenými klíčovými slovy SYMBOL ... END, CONST ... END, VAR ... END

!! Vzhledem ke změně způsobu využívání paměti pro proměnné je zrušena funkce „zásobníku“. Zápis do proměnné STACK přepisuje tuto a pouze tuto proměnnou nezávisle na hodnotě POINTER.

8.4.1 Otazníková deklarace

V předchozí verzi SIMPLE byla zavedena deklarace proměnných v adresovém prostoru předdefinovaných proměnných, která nevyžadovala od programátora přesný výběr adresy, resp. ztotožnění s konkrétní předdefinovanou proměnnou. Syntaxe tohoto způsobu deklarace byla zachována pouze z důvodů zpětné kompatibility, považuje se však za neperspektivní a nebude v budoucnu podporována. proto ani nebudeme uvádět její syntaxi. Konverzní program ji ještě využívá pro možnost překladu starších programů, vy se jí však raději vyhněte.