KIT-BUILDER

Jak na to!?

Střešovická 49, 162 00 Praha 6, e-mail: sofcon@sofcon.cz
tel./fax.: (02) 20 61 03 48 / (02) 20 180 454, http:// www.sofcon.cz

SofCon®s.r.o. 23.11.1999 *Ver. 3.03*

Obsah :

1. Úvod **1**

2. KIT-BUILDER v „kostce“ **2**

3. Postup vykonávání programu **3**

3.1 Úvod 3
3.2 Hlavní cykl – proces MAIN 3
3.3 Přerušení – proces FAST 3
3.4 Ostatní procesy 3

4. Tvorba aplikace od A do Z **4**

4.1 Úvod 4
Příklad: START *4*
4.2 Návrh projektu (programu), program KBDLCD 4
4.3 Kontrola a překlad programu 4
4.4 Simulace programu na PC 5
4.5 Nahrání programu do řídící jednotky, terminálu - program KBDCON 5
4.6 Verze EPROM 6
4.6.1 KITV40 (KIT386EX), TERM01 6
4.6.2 TERM10A 6
4.6.2.1 Obrazovka SETUP TERMINÁL TERM10 6
4.6.2.2 Obrazovky KIT BUILDER NASTAVENÍ 6
4.6.3 TERM10B 6
4.7 Zajištění spustitelnosti, běhu komunikace 6
4.8 Test propojky PBUS 6

5. Uživatelská datová struktura (registry) **8**

5.1 Celočíselné registry 8
5.2 Proměnné typu string 8
5.3 Reálné registry 9
5.4 Časovače (Timer) 9
5.5 Systémové registry 9
5.6 Pole 9
5.7 Informace o využití datové struktury 10

6. Jazyk KIT-BASIC **11**

6.1 Úvod - Struktura zdrojového programu 11
6.2 Všeobecná pravidla pro zápis programu 11
6.3 Komentáře 11
6.4 Deklarace symbolických konstant 11
6.5 Deklarace symbolických jmen registrů 12
6.6 Deklarace symbolických proměnných s automatickým přiřazením registru 12
6.7 Přetypování 13
6.8 Deklarace HW-objektů a SW-objektů 14
6.9 Deklarace procedur (vlastní řídící algoritmus) 14
6.10 Procedura MAIN 14
6.11 Procedura FAST 15
6.12 Uživatelská procedura 16
6.13 Příkazy a operace 16
6.13.1 Přiřazovací příkaz 16
6.13.2 Příkaz volání procedury 17
6.13.3 Složený příkaz 17
6.13.4 Podmíněný příkaz 17
6.13.5 Příkaz větvení 17
6.13.6 Příkaz cyklu REPEAT 17
6.13.7 Příkaz cyklu WHILE 18
6.13.8 Příkaz cyklu FOR 18
6.13.9 Příkaz ukončení procedury 18

6.13.10 Příkaz MainExit 18
6.13.11 Příkaz WAIT 18
6.13.12 Příkaz obecného skoku 18
6.13.13 Aritmetické operace 19
6.13.14 Logické operace 19
6.13.15 Operace relací 19
6.13.16 Operace adresa 19
6.13.17 Procedury inkrementace a dekrementace 19
6.13.18 Operace posunů 19
6.13.19 Goniometrické funkce 19
Příklady: DGONIO, DGONIOR *19*
6.13.20 Další matematické funkce 19
Příklad: DRND *20*
6.13.21 Funkce výpočtu kódů CRC 20
Příklad: DCRC *20*
6.13.22 Procedury pro práci s typem DateTime (časem) 20
Příklady: DHODINY, DHODINYG *20*
6.13.23 Procedura pro práci se stringy 21
6.13.24 Procedury přesunu – Move, MoveXXX 21
6.13.25 Procedury PROCi 21
6.13.26 Procedura obsluhy tiskárny 21
Příklad: DLPT *21*
6.13.27 Procedury obsluhy displeje řady LED 21
Příklad: DLEDKIT *22*
6.14 Vyhodnocování výrazů 22
6.15 Ošetření chyb při vyhodnocování výrazů 24
6.16 Systémové definice pro lexikální analyzátor 24
6.16.1 definice #define 24
6.16.2 definice #include 24
6.17 Systémové definice - options 25
6.17.1 Definice ProgVer 25
6.17.2 Definice FastFreq 25

7. Terminály a tvorba obrazovek **26**

7.1 Úvod 26
7.2 Terminál řady TERM10 26
Příklady: DTEXT, DGRAF *26*
7.3 Terminál TERM01 26
Příklad: DTERM01 *27*
7.4 Generátor stringů GENSTR 27
7.5 Prvky na obrazovce 27
7.5.1 Úvod 27
7.5.2 Popis BITMAP 28
Příklad: DGRAF *28*
7.5.3 Popis FONT 28
Příklad: DTEXT *28*
7.5.4 Popis POSITION 28
Příklad: DTEXT *28*
7.5.5 Popis PRINT 29
Příklad: DTEXT *29*
7.5.6 Popis EDIT 29
Příklad: DEDIT *30*
7.5.7 Popis EDITP 30
Příklad: DEDITP *30*
7.5.8 Popis EDITENTER 30
Příklady: DEDIT, DEDITP *30*
7.5.9 Popis RECT 30
Příklad: DGRAF *30*
7.5.10 Popis FILL 30
Příklad: DGRAF *30*

7.5.11 Popis LINE 30
Příklad: DGRAF *31*
7.5.12 Popis POINT 31
Příklad: DGRAF *31*
7.5.13 Popis CIRCLE 31
Příklad: DGRAF *31*
7.5.14 Popis GRAPH 31
Příklad: DGRAPH *32*
7.5.15 Popis GRAPHXY 32
Příklad: DGRAPHXY *33*
7.5.16 Popis BAR 33
Příklad: DBAR *33*
7.5.17 Popis ONKEY (reakce na stisk kláves) 34
Příklad: DONKEY *34*
7.5.18 Popis WAIT 34
Příklad: DWAIT *34*
7.5.19 Popis CASE 34
7.5.20 Popis obrazovky HELP 35
Příklad: DTEXT *35*
7.6 Použití prvků pro jednotlivé typy terminálů 35

8. Vstupy a výstupy **38**

8.1 Úvod 38
8.2 Typ IOPBUS 38
Příklad: DIOPBUS *39*
8.3 Typ IODIO01 39
Příklad: DIODIO01 *39*
8.4 Typ IODOO01 39
Příklad: DIODOO01 *39*
8.5 Typ IODXO01 39
Příklad: DIODXO01 *39*
8.6 Typ IOTERM10 40
Příklad: DIOT10 *40*
8.7 Typ IODTERM10 40
8.8 Typ IOATERM10 40
8.9 Typ IOADDA01 41
Příklad: DIOADDA *42*
8.10 Typ IOFLEXPOS 42
Příklad: DIOFLEXP *43*
8.11 Seznam HW desek a jejich ovladačů 44

9. Komunikační linky **45**

9.1 Úvod 45
9.2 Konfigurace 45
9.3 Procedury 45
Příklady: DCOM, DCOMPRT *46*
9.4 Parametry komunikačních protokolů 47
9.4.1 Úvod 47
9.4.2 COM 47
9.4.3 COMBR 47
9.4.4 COMPB 47
9.4.5 PRT 48
9.4.6 TECOM 48
9.4.7 SAIA 48
9.4.8 LECOM – připravuje se 49
9.4.9 Rockwell Automation (Allen-Bradley) 49

10. Speciální SW objekty **50**

10.1 Úvod 50
10.2 SW-objekt TAB 50
10.3 SW-objekt SAVER 51

Příklad: DSAVER *51*
10.4 SW-objekt ARCHIV 51
Příklad: DARCHIVE *53*
10.5 SW-objekt PID regulátor 53
Příklad: DPID *54*
10.6 SW-objekt PIDR regulátor 54

11. Seznam systémových registrů **56**

12. Příklady **57**

12.1 Program DPRVNI 57
12.2 Abecední seznam demo příkladů 58

13. Otázky a odpovědi **59**

1. Úvod

Cílem této příručky je popsat a ukázat možnosti tvorby projektu ve vývojovém prostředí **KIT-BUILDER**.

V kapitole 2 se seznámíme s celkovou filosofií projektu **KIT-BUILDER** a programovacího jazyka **KIT-BASIC**, v kapitole 3 si popíšeme způsob zpracování jednotlivých částí programu při běhu aplikace.

V kapitole 4 najdeme popsány jednotlivé fáze vývoje celé aplikace v prostředí **KIT-BUILDER** doplněné o jeho jednotlivé moduly (KBDLCD, KBDCOMP, KBPROC01, KBPROC10, KBDCON) a jednotlivé modifikace obsahu EPROM dle standardní hardwarové konfigurace.

V kapitole 5 se seznámíme s uživatelským paměťovým prostorem a se způsobem adresování, v kapitole 6 si postupně popíšeme strukturu, základní operace, procedury a funkce jazyka KIT-BASIC. Kapitola 7 je věnována obsluze TERMINÁLŮ a popisu jednotlivých prvků pro návrh jejich obrazovek.

V kapitole 8 se seznámíme s ošetřením jednotlivých vstupů a výstupů, v kapitole 9 s konfigurací a obsluhou jednotlivých komunikačních linek.

V kapitole 10 je uveden popis speciálních prvků typu PID regulátor, Archívy, Tabulky atd. .

V kapitole 11 najdeme seznam všech systémových registrů.

V kapitole 12 najdeme vedle popisu základního demo příkladu seznam dalších demo příkladů umístěných na instalačních disketách.

V kapitole 13 najdeme odpovědi na nejčastější otázky které nám zákazníci kladou.

2. KIT-BUILDER v „kostce“

KIT-BUILDER je vývojové prostředí, určené pro naprogramování a odladění řídících systémů, postavených na bázi řídících jednotek **KITV40** příp. **KIT386EX**, průmyslového terminálu **TERM10** nebo pro programování kompaktních řídících systémů **KOMPAKT**. Základem programového prostředí **KIT-BUILDER** je překladač jazyka **KIT-BASIC** a interpret **KIT-PROCESSOR**. Programovací jazyk **KIT-BASIC** vychází ze zjednodušené struktury jazyka Pascal, resp. Basic, a je doplněn o prvky z jazyků programovatelných automatů a speciální prvky poplatné aplikacím v regulaci, MaR atd. . **KIT-PROCESSOR** umožňuje běh přeloženého uživatelského programu jak v ladícím prostředí na personálním počítači, tak v originálním prostředí **KITV40**, **KIT386EX TERM10** nebo **KOMPAKT**. O jednotlivých modulech se více dočtete v kapitole 4.

Uživatelský program zapsaný v jazyce **KIT-BASIC** podrobně popisuje žádaný algoritmus, uvedený v procedůrách **INIT**, **MAIN** a **FAST**. Dále obsahuje popis jednotlivých obrazovek procesu **TERMINAL** včetně algoritmů, které se mají provést jako reakce na stisk kláves. Kromě toho jsou v jazyce obsaženy deklarace popisující konfiguraci připojeného hardware (tzv. **HW-objekty**), deklarace dalších paralelně běžících procesů (tzv. **SW-objekty**). Konečně je možno deklarovat symbolické názvy pro jednotlivé registry i absolutní konstanty.

Pro uživatele poskytuje jazyk **KIT-BASIC** dvě datové struktury - uživatelské registry typu real a celočíselné uživatelské registry typu bit, byte, word, integer, longint a datetime, které sdílejí jeden adresový prostor, nad kterým se překrývají. Speciálním pohledem do oblasti celočíselných registrů je pak registr typu string. Všechny registry mohou být definovány absolutně svojí adresou, nebo symbolicky.

Mezi **HW-objekty** patří popisy všech desek stavebnice **KIT**, které je možno využít při stavbě řídícího systému. Pro každou vstupně/výstupní desku jsou definovány její hardwarové adresy, sada uživatelských registrů, pomocí kterých se ovládají vstupní, resp. výstupní signály a časový režim ovládání (**MAIN** resp. **FAST**).

Mezi **SW-objekty** patří autonomně běžící časovače, filtry, čítače, generátory pulsů, archivátory, PID-regulátory, hodiny reálného času, obsluha komunikačníhch kanálů atd. Tyto objekty sledují přiřazené hardwarové signály a do přiřazených uživatelských registrů předávají již komplexní předzpracované informace.

Pro popis vlastních algoritmů jsou zavedeny standardní příkazy vyskytující se v Pascalu, resp. Basicu (if-then-else, case-of-else-end, repeat-until, while-do, for-to/downto-do), příkaz volání procedury. Jednotlivé příkazy jsou vysvětleny v dalších kapitolách tohoto dokumentu. Pro uložení nové hodnoty do registru se používá klasický přiřazovací příkaz (= resp. :=), ve výrazech je možno používat všechny běžné relační, logické a aritmetické operátory (<, >, <=, >=, <>, =, and, or, xor, not, shl, shr, +, -, *, /). Konverze mezi jednotlivými typy registrů se provádí automaticky. Jsou definovány běžné reálné funkce (sin, cos, sqrt(odmocnina) atd.).

Ve vlastním uživatelském programu (proceduře **MAIN**) se nezapisují žádné instrukce vstupů/výstupů - veškerá komunikace s reálným prostředím se děje pomocí **HW-objektů**. Procedura **MAIN** na svém počátku předpokládá, že jsou všechny vstupy aktuálně přečteny do příslušných registrů a na svém konci předpokládá, že všechny registry, použité ve výstupních objektech budou zapsány na výstupní signály.

Pro popis obrazovek slouží deklarace **TERMINAL**. V těchto deklaracích jsou popsány jednotlivé obrazovky, např. terminálu **TERM10**. Je možno definovat více obrazovek, každou buď v módu zobrazení, nebo v módu editace.

U každé obrazovky v **módu zobrazení** je možno definovat:

1. číslo podkladové bitmapy,
2. množinu grafických objektů, které mají být na obrazovce zobrazeny,
3. zobrazený text včetně čísel použitých fontů a pozice textu na obrazovce,
4. algoritmus, který se má provést při stisku jednotlivých kláves.

Jako text může být vypisován:

1. zadaný pevný text,
2. jeden ze zadaných pevných textů v závislosti na obsahu určité uživatelské proměnné,
3. různě naformátované obsahy uživatelských registrů.

V **módu editace** je možno na obrazovce vypsat současnou hodnotu uživatelského registru, po jejím případném zeditování zkontrolovat, jestli je nová hodnota v přípustných mezích a v případě kladného výsledku provést zápis do uživatelského registru.

Ke každé obrazovce je možno nadefinovat HELP-obrazovku, na které může být popsán způsob obsluhy původní obrazovky.

3. Postup vykonávání programu

3.1 Úvod

Zpracování programu při běhu realné aplikace v prostředí KIT-BUILDER je založeno na několika základních „paralelně“ běžících procesech popsaných dále v této kapitole.

3.2 Hlavní cykl – proces MAIN

Postup vykonávání hlavního procesu - uživatelského programu, napsaného v jazyku KIT-BASIC - je založen na principu činnosti programovatelných automatů. Cyklicky se provádí:

1. Obsluha WatchDogu.
2. Obsluha Hodin.
3. Čtení hodnot ze vstupních portů do uživatelem definovaných registrů v seznamu MAIN.
4. Vykonávání všech instrukcí z uživatelem definované procedury (procesu) MAIN. Zpracování může být přerušeno zavoláním příkazu WAIT, při dalším cyklu se bude pokračovat na následující instrukci za příkazem WAIT.
5. Zpracování reakce na stisknutou klávesu terminálu, obsluha LED terminálu (např. TERM10).
6. Zápis na výstupní porty z uživatelem definovaných registrů v seznamu MAIN.
7. Obsluha jedné položky vstupů a výstupů ze seznamu CASE.
8. Uložení zálohovaných registrů – seznam Save.
9. Akce prováděné jednou za sekundu – časovače 1s , obsluha archívů od časovače.
10. Systémová obsluha procesu MAIN.

Pokud je celý cykl delší než cca 2s, nedojde k včasné obsluze WatchDogu a je proveden RESET systému. Program je dále zpracováván znovu od počátku, včetně volání uživatelské procedury INIT.

3.3 Přerušení – proces FAST

Pravidelně pod přerušením každých 10 (20, 50) ms je prováděna obsluha následujícího cyklu:

1. Čtení hodnot ze vstupních portů do uživatelem definovaných registrů v seznamu FAST.
2. Obsluha časovačů 10ms.
3. Vykonávání všech instrukcí z uživatelem definované procedury (procesu) FAST – pouze pokud je definována.
4. Zápis na výstupní porty z uživatelem definovaných registrů v seznamu FAST.
5. Systémová obsluha procesu FAST.

3.4 Ostatní procesy

Na pozadí „paralelně“ se zpracováním hlavního cyklu a obsluhy procesu FAST probíhají následující procesy:

1. Obsluha terminálů (např. TERM10, GENSTR).
2. Obsluha regulátorů (např. PID).
3. Obsluha komunikačních linek (např. COM).

4. Tvorba aplikace od A do Z

4.1 Úvod

V této kapitole jsou popsány jednotlivé fáze vývoje aplikace v prostředí KIT-BUILDER, resp. základní programové moduly určené pro tvorbu a ladění celého projektu. Jednotlivé fáze jsou postupně ilustrovány na příkladu s názvem „START“.

Příklad: START

Kapitola je členěna podle jednotlivých fází tvorby aplikace do následujících kapitol:

1. Návrh projektu (programu).
2. Kontrola a překlad programu.
3. Simulace programu na PC.
4. Nahrání programu do řídící jednotky, terminálu - program KBDCON.
5. Verze EPROM.

4.2 Návrh projektu (programu), program KBDLCD

Kompletní návrh projektu (programu) je možno provádět na dvou základních platformách.

První varianta spočívá ve využití programu KBDLCD, který na bázi grafického prostředí doplněného o textové prvky umožňuje iterativní a jednoduchou tvorbu vašeho projektu.

Druhá varianta spočívá ve vytvoření celého projektu v libovolném textovém editoru, který do textu nedoplňuje speciální formátovací znaky. Jmenujme např. program EDIT.COM (DOS), interní editor vašeho správce souborů či NOTEPAD.EXE (WIN). Výstupem vaší tvorby musí být soubor zakončený příponou PRG, případně doplněný o tzv. include soubory zakončené příponou PRI, např.: RERULACE.PRG, TERM.PRI.

Pro názornost si uveďme příklad s programem START. V programu KBDLCD v položce „Soubory | Otevři“, najdeme projekt „START.LCD“. Otevřeme tak projekt, sestávající z jedné obrazovky terminálu TERM, jménem „HOUKACKA“, a souboru START.PRG definujícího programovou a konfigurační část projektu. Jednotlivé části si lze prohlédnout v položce „Program | Otevření okénka *.PRG” nebo „Program | Otevření okénka *.PRI”. Při otevírání druhého okénka může dojít k otevření prázdného okna, což je způsobeno tím, že daný soubor nebyl ještě vygenerován (generován programem KBDLCD, položka „Operace | Generování *.PŘI“).

4.3 Kontrola a překlad programu

Vytvořený projekt (program) je třeba zkontrolovat a přeložit do binárního formátu (tzv. p-kódu), kterému rozumí samotný simulátor a interpret nahraný v paměti řídící jednotky či terminálu. Tento překlad se provádí programem KBDCOMP.EXE. V prostředí KBDLCD lze kompilátor volat přímo přes tlačítko v panelu nástrojů.

Při překladu uživatelského programu překladač "zastaví" překlad při nalezení první syntaktické nebo sémantické chyby a vypíše chybové hlášení, v kterém chybu pojmenuje. Po opravě této chyby ve zdrojovém textu může uživatel překlad opakovat.

Proběhne-li překlad bez chyb, vygeneruje se přeložený binární kód programu (tzv. p-kód) do souboru s příponou BIN, v našem případě START.BIN.

Spuštění programu KBDCOMP má následující syntaxi:

KBDCOMP JménoProgramu [parametr1..N]

Význam jednotlivých parametrů ukazuje následující tabulka:

parametr	význam
-m	Vygeneruje MEM-souboru obsahující tabulka všech použitých registrů a jejich symbolických názvů, včetně jejich rozložení v paměti. Soubor má příponu *.MEM.
-l	Lze použít pouze s parametrem –m. Vygeneruje tzv. "dlouhou" verzi MEM-souboru, tj. v souboru nejsou přeskakovány nevyužité registry a je vypsán komplexní přehled paměťové náročnosti aplikace. Soubor má opět příponu *.MEM.
-k	Vygeneruje tzv. MAP-souboru obsahující tabulku všech použitých registrů a jejich symbolických názvů určenou pro další zpracování v programu KBDCON. Soubor má příponu *.MAP.
-p	Vygeneruje textový soubor obsahující symbolický výpis p-kódu. Soubor má příponu *.PCD.
-s	Vygeneruje textový soubor obsahující symbolický výpis popisující obsluhu terminálu. Soubor má příponu *.STR.
-r	Okno překladače se po skončení překladu samo uzavře

příklad: příkaz KBDCOMP START -p -m

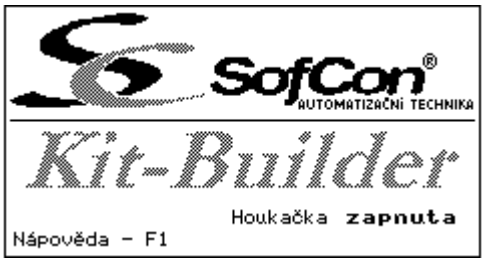
způsobí vygenerování souborů:

START.BIN - přeložený program

START.PCD - symbolická verze p-kódu

START.MEM - tabulka použitých registrů

4.4 Simulace programu na PC



Pro jednoduché otestování funkčnosti vytvořené aplikace jsou určeny programy KBPROC01 a KBPROC10. Programy jsou spustitelné pouze na platformě DOS a umožňují spuštění v následujících režimech:

pro reálný běh: KBPROCxx XXX –r

pro simulaci terminálu TERM10: KBPROC10 XXX –t

pro simulaci terminálu TERM01: KBPROC01 XXX –s

kde XXX je jméno přeloženého resp. zdrojového programu.

Při reálném běhu běží program na procesoru vašeho počítače, všechna HW zařízení lze připojit a obsluhovat přes sběrnici IO-Bus na desce PC-KIT, kterou můžeme umístit do ISA slotu vašeho počítače.

Při simulaci terminálu TERM10, resp. TERM01, se terminál TERM10, resp. TERM01, simuluje na obrazovce počítače, ostatní HW zařízení jsou obsluhována jako v reálném běhu (Klávesa Start je simulována klávesou Alt-A, klávesa Stop je simulována klávesou Alt-S, soutisk kláves SHIFT-ENTER soutiskem kláves CTRL-ENTER).

Programy KBPROC01 a KBPROC10 při běhu na PC lze kdykoliv ukončit soutiskem kláves Alt-X. Po ukončení zůstanou na disku PC vytvořeny soubory *.SAV. Tyto soubory simulují zálohovanou paměť RAM na reálném HW a mají tento význam:

ERROR.SAV - archív systémových chyb (po jeho smazání nelze číst staré záznamy o chybách)

SAVE.SAV - obsah registrů, uchovaných pomocí SW-objektu SAVER (po jeho smazání se chová program jako po prvním nahrání, kdy nejsou žádné hodnoty uschovány)

ARCHIVE.SAV - obsah oblasti, do které se ukládají data pomocí SW-objektů ARCHIVE (po jeho smazání se chová program jako po prvním nahrání, kdy nejsou žádné hodnoty uschovány)

GLB.SAV, GLB1.SAV - globální data doplněná CRC kódem zajišťujícím kontrolu jejich neporušenosti (po jeho smazání je nutno nahrát nový uživatelský program i fonty)

ALLOC.SAV, DATA.SAV - obsah banků programu, fontů a bitmap (po jejich smazání je nutno nahrát nový uživatelský program i fonty)

Pozn.: Pokud v reálném běhu neuvedeme jméno programu, probíhá inicializace programu ze zálohovaných struktur.

POZOR - programy KBPROC01.EXE a KBPROC10.EXE vyžadují velkou základní paměť, proto je nelze spustit, máme-li v CONFIG.SYS resp. AUTOEXEC.BAT, nadefinováno spuštění většího množství nebo rozsáhlejších ovladačů nebo rezidentních programů!

V programu KBDLCD lze opět simulátor volat přímo přes tlačítko v panelu nástrojů. V případě demo programu START se nám otevře následující okno se spuštěnou aplikaci START.

Pozn.: Při simulaci na PC se aplikace ukončuje stiskem Alt-X, tlačítku STOP odpovídá kombinace kláves Alt-S a tlačítku START Alt-A na klávesnici vašeho PC.

4.5 Nahrání programu do řídící jednotky, terminálu - program KBDCON

Program KBDCON je určen ke komunikaci s reálnou řídící jednotkou, terminálem (dále jen řídící jednotka).

Program KBDCON umožňuje sestavení jednotlivých modulů aplikace (program, fonty, bitmapy) v jeden celek, jejich nahrání do řídící jednotky *(pouze při zastaveném aplikačním programu v řídící jednotce)*, sledování a modifikaci paměťových registrů, čtení obsahu archívů, zastavování a spouštění programu v řídící jednotce. Program dále umožňuje spouštět samotný překladač KBDCOMP, prohlížet si nahrávané bitmapy a fonty.

Komunikace programu KBDCON s řídícím systémem probíhá po sériové lince, zpravidla COM2. Správnou editací INI souboru lze nastavit komunikaci na jinou komunikační linku. S řídícím systémem lze komunikovat i přes telefonní modem (pevná linka, GSM, radio-modem). Tento požadavek musí být uveden při objednávání řídícího systému.

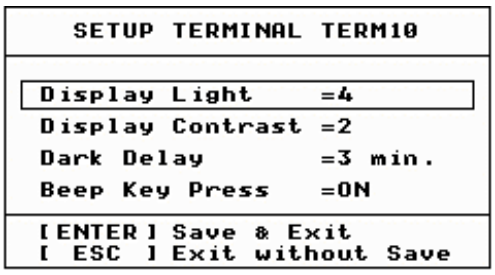
Podrobnější popis jednotlivých funkcí programu naleznete po jeho spuštění přímo v nápovědě.

4.6 Verze EPROM

V této kapitole jsou popsány jednotlivé implementace PROCESORU ve verzi EPROM v závislosti na standardním hardwaru.

4.6.1 KITV40 (KIT386EX), TERM01

Sestava je založena na řídící jednotce KITV40 (KIT386EX) s možností připojení terminálu TERM01 na sériový kanál. Do této jednotky lze tedy nahrát program ber definice terminálu, nebo s definicí terminálu TERM01.

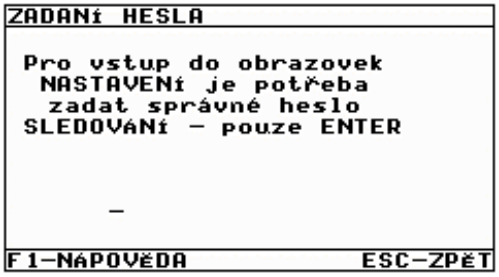


4.6.2 TERM10A

Sestava je založena na řídící jednotce KITV40 (KIT386EX) doplněné terminálem TERM10A. Bázová adresa terminálu je pevně dána $2300.

4.6.2.1 Obrazovka SETUP TERMINÁL TERM10

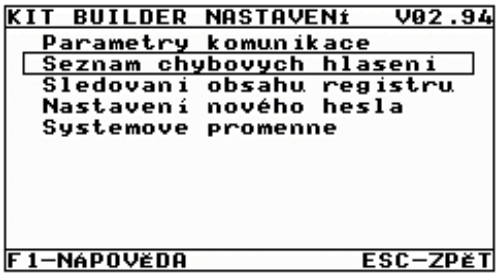
Pomocí stisku kombinace kláves SHIFT-ENTER se dostaneme do standardní SETUP obrazovky terminálu TERM10, kde můžeme nastavit jas, kontrast, dobu po které má terminál sám zhasnout a zda má terminál při stisku tlačítka vydávat zvukový signál. Mezi jednotlivými obrazovkami přeskakujeme klávesami šipka nahoru, resp. šipka dolu, rozsahy prohlížených registrů volíme klávesami šipka doleva, resp. šipka doprava. Zpět se dostaneme stiskem klávesy ESC, resp. ENTER.



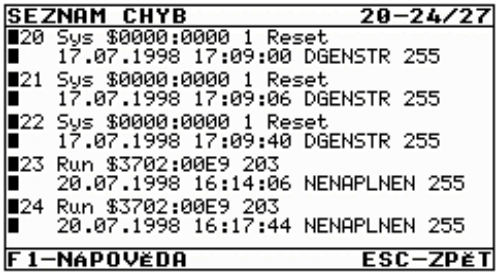
4.6.2.2 Obrazovky KIT BUILDER NASTAVENÍ

Pomocí stisku klávesy F10 se dostaneme do obrazovky ZADÁNÍ HESLA.

Pokud nezadáme heslo a pokračujeme klávesou ENTER dostaneme se do obrazovek SLEDOVÁNÍ. Zde můžeme provádět sledování parametrů systémové komunikace, sledovat čas a datum, prohlížet registry, sledovat výpisy chybových hlášení atd. Pokud zadáme správné heslo, implicitní heslo je SOFCON, dostaneme se do obrazovek NASTAVENÍ. V těchto obrazovkách můžeme dělat vše jako v obrazovkách SLEDOVÁNÍ a navíc můžeme některé parametry nastavovat.

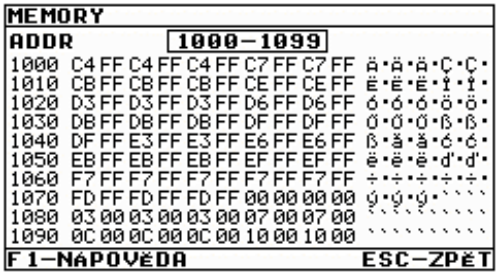


Na obrazovce NASTAVENÍ si můžeme vybrat jednu z položek, kterou chceme nastavovat nebo prohlížet. Např. obrazovka Seznam chybových hlášení je zobrazen na dalším obrázku.



Na obrazovce Parametry komunikace zase můžeme nastavit rychlost komunikační linky s PC, adresu(NODE) této stanice při systémové komunikaci po sériovém kanálu s PC, či nastavit parametry modemu, je-li nastaveno jeho použití.

Obrazovky SLEDOVÁNÍ OBSAHU REGISTRŮ nám umožňují sledovat současný stav obsahu registrů procesoru při běhu programu.



4.6.3 TERM10B

Obdobná sestava jako TERM10A, ale založená na terminálu TERM10B. Vše platí stejně jako pro TERM10A viz. kapitola 4.6.2.

4.7 Zajištění spustitelnosti, běhu komunikace

Po nahrání programu do řídící jednotky je jednotka vybavena detekcí soustavného resetu. Pokud systém v posledních 15 minutách více než 10x prošel chybou s resetem, dojde k zastavení programu. Programátor tak má možnost navázat se systémem komunikaci, případně si přečíst kód chyby, ke kterému dochází. Pokud nedojde do cca. 10 minut k navázání systémové komunikace, je systém opět resetován a pokouší se znovu spustit program.

4.8 Test propojky PBUS

Zejména pro odlaďování je vhodné mít pojistku, kterou zajistíme, aby nedocházelo k opakovanému spušťení programu. První pojistka je popsána v předchozím odstavci, druhou, trochu robustnější si popíšeme nyní.

Procesory V40, i I386SX jsou vybaveny tzv. sběrnicí PBUS. Ve většině aplikací se tato sběrnice nevyužívá a zůstává tudíž prázdná. Toho je využito pro následující funkci. Pokud na konfigurační adresu 4067.0 pomocí programu KbdCon zapíšeme hodnotu 1, aktivujeme funkci test propojky PBUS při startu systému. Pokud je propojka v poloze 47-49, systém nelze odstartovat, pokud je v poloze 47-48, start programu je povolen. Potlačení testovací funkce provedeme nastavením hodnoty 0 na výše jmenované adrese.

Pokud je tento port v aplikaci využit, je nutno funkci test propojky vypnout.

5. Uživatelská datová struktura (registry)

Základem aplikace v prostředí KIT-BUILDER je paměťový prostor (registry), nad kterým je celý program vystaven (výpočty, obrazy vstupů a výstupů, řídící registry pro terminály, komunikační linky atd.).

V prostředí KIT-BASIC jsou deklarovány dvě základní banky registrů, celočíselné registry a registry s pohyblivou desetinnou tečkou, dále jen reálné.

Velikost banky uživatelsky přístupných celočíselných registrů je 4000 Byte, velikost banky reálných registrů 250 reálných čísel, kde pod pojmem reálné číslo rozumíme číslo uložené ve struktuře 6 Byte.

Veškeré operace vstupů a výstupů, časovačů či čítačů, regulátorů, terminálů a jiných objektů jsou vykonávány nad těmito univerzálními registry. Výjimku tvoří objekty archívů, které pracují nad další pamětí velikosti cca 60kB.

Specialitou celočíselných registrů je možnost pracovat s nimi jako s registry typu bit (0 nebo 1), byte (interval 0 až 255), word (interval 0 až 65535), integer (interval -32768 až 32767), longint (interval -2147483648 až 2147483647), string (byte délky + 1 až 255 znaků) a datetime, což je speciální typ pro uložení datumu a času v komprimovaném formátu.

Speciální skupinou celočíselných registrů jsou registry systémové, které jsou umístěny mimo oblast uživatelských registrů a jsou přístupné přes symbolická jména přiřazená k těmto registrům (viz kap. 11).

5.1 Celočíselné registry

Jak již bylo řečeno, celočíselné registry je možno adresovat jako bit, byte, word, integer, longint a datetime (registr string je popsán speciálně v kapitole 5.2). Adresování provádíme uvedením prvního klíčového písmene a doplněním čísla základního registru v rozsahu 0 až 3999, např. B4, W8, I156, L1000, D1600 (výjimku tvoří bit - písmeno „B" nebo „W“ se doplňuje symbolem "." a číslem bitu, např. B7.3, W6.10). Pod pojmem základní registr rozumíme adresu registru počítanou v bytech. Vše je patrné z následující tabulky popisující překrývání registrů v paměti.

B0	B1	B2	B3	B4	B5	B6	B7	Byte
W0		W2		W4		W6		Word
I0		I2		I4		I6		Integer
L0				L4				LongInt
D0				D4				DateTime

B0.7	B0.6	B0.5	B0.4	B0.3	B0.2	B0.1	B0.0	Bit
B0								Byte

Jak je z tabulek patrné, adresa registrů W a I musí být násobkem dvou a adresa registru L a D násobkem čtyř. Pokud není toto pravidlo zachováno, překladač hlásí chybu.

Poznámka: vícebytové proměnné mají vždy uložen nejnižší byte na nejnižší adrese.

5.2 Proměnné typu string

Do prostředí celočíselných registrů jsou dále adresovány i stringové proměnné. Stringovou proměnnou značíme písmenem "S", doplněným o číslo bytu, na kterém začíná. Může to být libovolné číslo bytu, např. S100.

Stringová proměnná má implicitně velikost 20 bytů, pokud v sekci SYMBOL nenadefinujeme jinak (minimální přípustná velikost je 2 byty, maximální přípustná velikost je 256 bytů). V prvním bytu každé stringové proměnné je uložen aktuální počet platných znaků (tj. délka stringu), od druhého bytu je uložen po znacích daný string. Ve stringové proměnné o velikosti x bytů může být uložen string o maximální délce x-1 znaků, tj. ve stringové proměnné implicitní délky 20 bytů může být uložen string, skládající se maximálně z 19 znaků. Zbylé byty stringové proměnné do její celkové délky mají nedefinovaný obsah.

Celkovou délku stringové proměnné zjistíme pomocí standardní funkce SIZEOF(Sx), počet aktuálně platných bytů standardní funkcí LENGTH(Sx) nebo přečtením bytu Bx, kde je aktuální počet platných bytů stringové proměnné Sx uložen

Jako příklad lze uvést proměnnou S4, která má implicitní velikost 20 bytů a počet právě platných znaků 15. Tato proměnná je v adresovém prostoru uložena takto:

B3	B4	B5	B6	B7	...	B19	B20	B21	B22	B23	B24	
	S4	prostor vyhrazený datové části proměnné S4										
	15	"A"	"H"	"O"	...	"Z"	??	??	??	??		

Jednotlivé znaky stringové proměnné můžeme výjimečně číst pomocí registrů typu byte - na výše uvedeném obrázku B5 až B23.

Na stringové proměnné můžeme aplikovat relace <> a =, operace přiřazení := a ADDS pro spojování stringů (podrobněji kapitola 6.13.23).

Při vyhodnocování těchto operací se provádí automatická kontrola maximální délky cílového registru tak, aby nedošlo k překročení vymezené oblasti délky 20 bytů, resp. uživatelem určené délky v bytech při deklaraci stringu v sekci symbol.

Vedle těchto základních operací můžeme s registry typu string pracovat v objektech typu TERMINAL, při kódování a dekódování zpráv z komunikačních kanálů atd. .

5.3 Reálné registry

Adresa u reálných registrů se skládá z písmene R následovaného číslem registru, např. R1, R111 atd. .

5.4 Časovače (Timer)

Registry typu word mohou být použity ve speciální funkci časovače. Základní perioda časovače je 10ms a 1s. Pokud je časovač spuštěn, provádí se pravidelně v zadaných časových okamžicích inkrementace obsahu použitého registru. Časovač se spouští speciální funkcí **TimerOn(reg,typ)**, kde parametr reg je adresa registru typu word, nad kterým bude časovač pracovat, parametr typ určuje základní periodu časovače (per10ms, per1s). Maximální počet časovačů spuštěných v jednom okamžiku je 256 od jednoho typu. Pokud již časovač nepotřebujeme, můžeme ho vypnout příkazem **TimerOff(reg)**, kde parametr reg je adresa registru typu word, nad kterým časovač pracuje. Pokud je časovač spuštěn s periodou 1s resp. 10ms a my ho spustíme s periodou 10ms, resp. 1s, provede se automaticky zrušení původní a nastavení nové periody časovače. Při spuštění ani při zastavení se hodnota registru reg nenuluje, vynulování provedeme jednoduchým přiřazovacím příkazem reg:=0. Maximální doba běhu časovače s krokem 10ms je 10minut, 66sec a 350ms, časovače s krokem 1s je 18hodin, 12minut a 15sec. Při naplnění časovače dojde k jeho zastavení. Registr použitý pro počítání obsahuje hodnotu 65535.

V následujícím příkladu se v proceduře MAIN testuje, zda uplynula doba více než 60 sec od příchodu poslední sestupné hrany pulzu na portu A desky IODIO01.

```
CONFIGURATION
 HWOBJ=IODIO01, ADR=$2300, MAIN=[INA]
CONST
 MezniDoba=60;
symbol
 T3=Word;
procedure INIT;
begin
 TimerOn(T3,per1s);
end;

procedure MAIN;
begin
 if IODIO01_A.0=1 then T3:=0;
 ...
 if T3>=MezniDoba then ...
end;
```

Poznámka: Použijeme-li option FastFreq (viz kap.6.17.2), rychlost inkrementace časovačů 10 ms se nemění, zůstává zachován poměr 100 tiků za 1 sec. ,ale přičítá se hodnota 2 při 20ms periodě, resp. 5 při 50ms periodě volání procesu FAST.

5.5 Systémové registry

Systém je vedle uživatelských registrů vybaven speciálními tzv. systémovými registry. V těchto registrech jsou uloženy informace o systému (např. aktuální čas a datum, jméno programu, verze procesoru a jiné). K těmto registrům je možno přistupovat pouze přes předdefinovaná symbolická jména jejichž přesný výčet včetně významu je uveden v kapitole 11.

5.6 Pole

Další možností prostředí KIT-BUILDER je řazení registrů do pole. Pole je řada registrů umístěných v paměti spojitě za sebou.

Adresování v poli se provádí uvedením jména pole, resp. adresy prvního prvku v poli, a jeho doplnění indexem umístěným v hranatých závorkách. Index vyjadřuje vždy offset v počtu bytů, např. B0[12] ve skutečnosti adresuje B12.

Obdobně se adresace provádí i u ostatních typů registrů. Adresa u registrů typu Word, Integer resp. LongInt musí být násobkem dvou, resp. čtyř. Toto pravidlo přechází i na indexy u polí těchto typů. Znamená to tedy, že indexy u pole typu Word a Integer musí nabývat pouze hodnot 0,2,4,…, u polí typu LongInt hodnot 0,4,8,… .

Index může být dán i obsahem registru, jehož adresa je umístěna v hranatých závorkách, např. B0[B1]. Nesmíme však zapomínat, že tento způsob adresování je náročnější na čas, adresace probíhá až 3 krát déle než u normálního adresování.

Upozornění:

1. Pole nemohou být tvořena z proměnných typu String.
2. Uživatel sám musí kontrolovat povolený rozsah indexu - překladač ani interpret kontrolu neprovádí.
3. Pozor při definicích v CONF sekci. Zde musí být vždy jasné o jakou adresu se jedná, nelze zde tedy používat na místě indexu proměnnou.

Příklad:

```
symbol
 Pole1=Byte:6;      {deklaruje pole typu Byte
                     o délce 6 Byte}
 Pole2=Integer:12; {deklaruje pole typu
                     Integer o délce 12 Byte
                     což představuje 6 Integer
                     čísel}
 Pole3=LongInt:24; {deklaruje pole typu
                     LongInt o délce 24 Byte
                     což představuje 6 LongInt
                     čísel}
 Index=Byte;        {deklaruje proměnnou typu
                     Byte }
…
Main
begin
…
 for Index:=0 to 6 do
 begin
 Pole3[Index*4]:=Pole1[Index]
                  *Pole2[Index*2];
               {provedeme součin prvků pole1
                s prvky pole2 a uložíme je
                do prvků pole3.}
 end;
…
end;
```

Poznámka: index může nabývat i záporných hodnot.

5.7 Informace o využití datové struktury

O tom, jak jsou využity jednotlivé registry uživatelské datové struktury, nám podává informace tzv. MEM-soubor, který je možno vygenerovat při překladu programu (viz kap.4.3). V tomto souboru jsou též zapsány veškeré zavedené symbolické názvy registrů - viz kap. 6.5 a 6.7.

6. Jazyk KIT-BASIC

6.1 Úvod - Struktura zdrojového programu

KIT-BASIC je nástroj pro popis celého projektu. Jednotlivé příkazy, funkce a jazykové konstrukce lze rozdělit do následujících základních skupin:

1. deklarace symbolických konstant
2. deklarace symbolických jmen registrů
3. deklarace autonomně pracujících HW-objektů a SW-objektů
4. popis řídícího algoritmu programu
5. deklarace jednotlivých obrazovek terminálu
6. deklarace uživatelských procedur

V programu může být nadefinováno libovolné množství uživatelských procedur (jednoúrovňově). Uživatelské procedury lze definovat pouze jako procedury bez parametrů.

Speciálními uživatelskými procedurami jsou procedury:

1. MAIN - algoritmus hlavního programu (tato procedura musí být vždy uvedena).
2. FAST - algoritmus, vykonávající se každých 10 (20, 50) ms.
3. INIT - algoritmus, vykonávající se po RESETu.

6.2 Všeobecná pravidla pro zápis programu

Program píšeme ve volném formátu, tj. nezáleží na rozložení zápisu programu do jednotlivých řádek. Každý jeden příkaz, resp. popis, je zpravidla ukončen středníkem. Začínajícímu programátoru se však doporučuje používat rozložení do jednotlivých řádek tak, jak je uvedeno v příkladech, tj. zpravidla jedna deklarace, resp. jeden příkaz, jazyka na jednom řádku.

V programu můžeme definovat uživatelské názvy pro konstanty, registry, SW-objekty, HW-objekty, procedury a návěští. Platí zásadní pravidlo, že každé symbolické jméno musí být nejdříve nadefinováno, a teprve později smí být použito. Z tohoto zásadního pravidla vyplývá, že jakoukoliv definici jména, pokud nevíme, co jméno znamená, musíme hledat při prohlížení zdrojového programu směrem "nahoru", tj. směrem k počátku textu.

Jednotlivé prvky jazyka pojmenováváme pomocí identifikátorů. Identifikátor se skládá z libovolného počtu písmen, číslic a znaků "\" a "_", avšak rozlišuje se pouze podle prvních 16 znaků.

Uživatel může vytvořit libovolný identifikátor, nesmí však definovat identifikátor stejného jména, jako je jedno z tzv. klíčových slov, což jsou identifikátory, nadefinované tvůrci systému. Seznam klíčových slov je uveden v příloze.

6.3 Komentáře

V programu můžeme používat na libovolných místech zápis komentáře, tj. libovolný text, zapsaný mezi komentářové závorky (takový text překladač ignoruje). Komentářové závorky jsou dvojího druhu - buď **{ }** nebo **(* *)**. K otevírací závorce patří uzavírací závorka z patřičné dvojice. Komentáře mohou být tímto způsobem i "vnořené": *(* .... {...} .... {...} .... *)* celý uvedený text je zakomentován.

6.4 Deklarace symbolických konstant

Pro přehlednost programu je možné používat symbolických jmen pro různé absolutní konstanty. Pokud toho chceme využívat, je třeba tato jména nadefinovat v sekci CONSTANT a pak je na kterémkoliv místě programu používat. Deklarace sekce CONSTANT může vypadat například takto:

```
constant
 PI=3.141592;  {definice konstanty typu real}
 MASKA=$FE;  {definice masky,zadane hexadec.}
```

Zapamatujte si základní pravidlo - jakýkoliv symbolický název musí být v programu nejprve nadefinován a teprve později smí být použit.

Ve výše uvedených příkladech jsou navíc použity komentáře, tj. libovolný text, zapsaný mezi komentářové závorky viz. kapitola 6.3.

6.5 Deklarace symbolických jmen registrů

Pro přehlednost programu je možné používat dále symbolických jmen pro jednotlivé registry z uživatelského datového prostoru (viz kapitola 4.8). Pokud toho chceme využívat, je třeba tato jména nadefinovat v sekci SYMBOL a pak je na kterémkoliv místě programu používat. Deklarace sekce SYMBOL může vypadat například takto:

```
symbol
 Vent1     =B0;       {odpovídá B0}
 Vent2     =B0+1;     {odpovídá B1}
 Vent3     =B0+2;     {odpovídá B2}
 PORT_A    =B20;      {odpovídá B20}
 TopeniZap=B20.3;    {odpovídá bitu c.3  B20}
 TopeniZap=PORT_A.3;{odpovídá bitu c.3 z B20}
 Vysledek =R13;      {odpovídá R13}
 Najeto    =W12;     {odpovídá W12, tj.B12+B13}
```

Tato symbolická jména registrů lze používat všude tam, kde jsou očekávána jména registrů.

Adresa registrů W a I musí být násobkem dvou a adresa registru L násobkem čtyř. Pokud není toto pravidlo zachováno, překladač hlásí chybu.

Při definování symbolického jména stringové proměnné můžeme definovat i její velikost:

```
symbol
 Jmeno1 = S3000;     {zabírá oblast B3000 až
                      B3019, max.počet znaků 19}
 Jmeno2 = S3040:40; {zabírá oblast B3040 až
                      B3089, max.počet znaků 39}
```

Při definování symbolického jména proměnné typu pole se postupuje jako u normálních proměnných, pouze se doplní za definici symbol dvojtečka „:“ a délka pole v Bytech u celočíselných registrů, v Registrech u realných registrů:

```
symbol
 Parametry =B0:10; {označuje B0-B9 jako
                    pole s názvem Parametry}
```

Poznámka: Pro každý typ registru jsou automaticky definována následující symbolická jména:

```
 B=B0;
 W=W0;
 I=I0;
 L=L0;
 D=D0;
 R=R0;
 S=S0;
```

Pokud tedy použijeme v programu proměnnou B, pracuje se automaticky s proměnnou B0, při použití W se pracuje s W0 atd. i pro ostatní typy.

6.6 Deklarace symbolických proměnných s automatickým přiřazením registru

Další významnou možností je definování symbolického jména proměnné daného typu bez uvedení konkrétní adresy registru. Překladač v tomto případě sám najde první volný registr příslušného typu (tj. pro typy byte resp. word, integer, longint první volnou skupinu bytů délky 1 resp. 2, 2, 4 v poli celočíselných registrů, pro typ real první volný registr typu real) a tomu přiřadí dané symbolické jméno. Uvedeme-li například definici

```
symbol
 A = byte;
 B = word;
 C = real;
 D = byte;
 E = byte;
 F = longint;
 G = string;
```

provede překladač následující přiřazení symbolických názvů daným registrům (za předpokladu, že nebyly dosud použity žádné registry):

```
symbol
 A=B0; {první volný byte v celočís.poli}
 B=W2; {první volný byte se sudým číslem v
        celočís. poli a za kterým je alespoň
        jeden byte volný}
 C=R0; {první volný real registr}
 D=B1; {první volný byte v celočís.poli}
 E=B4; {první volný byte v celočís.poli}
 F=L8; {první volný byte s číslem dělitelným
```

```
        4 v celočís.poli, za kterým jsou
        alespoň tři byte volné}
 G=S12;{první volný byte v celočís.poli, za
        kterým je alespoň 19 byte volných}
```

Techniku automatického přiřazení symbolů registrům můžeme použít i při definici SW a HW objektů, popsané podrobně dále v kapitole 6.7. V tomto případě můžeme například místo definice

```
CONFIGURATION
 HWOBJ=PBUS,NAME=KITPBUS,ADR=$d220,VAR=B500:3,
       MAIN=[INA,OUTC],C=$AA;
```

použít definici

```
CONFIGURATION
 HWOBJ=PBUS,NAME=KITPBUS,ADR=$d220,VAR=byte,
       MAIN=[INA,OUTC],C=$AA;
```

nebo

```
CONFIGURATION
 HWOBJ=PBUS,NAME=KITPBUS,ADR=$d220,MAIN=[INA,
       OUTC], C=$AA;
```

kde se parametr var doplní zcela automaticky.

Techniku automatického přiřazení symbolů **nelze** použít pro definici proměnných typu bit - ty musíme vždy definovat explicitně, při této definici však můžeme použít proměnnou typu byte nadefinovanou technikou automatického přiřazení symbolů, např. takto:

```
symbol
 VSTUP1=KITPBUS_A.6;
 VSTUP2=KITPBUS_A.7;
```

Zajímá-li uživatele, jak překladač přiřadil jednotlivé symboly jednotlivým registrům, je možné si prohlédnout MEM-soubor, generovaný překladačem viz. kapitola 4.3. Výše uvedená definice proměnných v sekci **symbol** a definice HW-objektu PBUS v sekci **configuration** má za následek vygenerování MAP-souboru (Mapa registrů) uvedeného v předchozím textu.

Za povšimnutí stojí, že překladač automaticky "vyplní volný prostor" na adr. B5 až 7 tím, že do ní umístí definici registrů, potřebných pro HW objekt PBUS.

Symbolická definice pole s automatickým přiřazením se definuje opět obdobně předchozímu. Při automatickém přiřazování se pouze provádí kontrola zda je v paměti dostatek volného místa pro nové pole, pokud není, hlásí chybu. Doporučujeme zkontrolovat MAP file, zda není možné některé proměnné přesunout a tak vytvořit dostatek paměti.

```
Mapa registrů
=============
Celociselne registry:
---------------------
$0000, 0000  A<B0>
$0001, 0001  D<B1>
$0002, 0002  B<W2+0>
$0003, 0003  B<W2+1>
$0004, 0004  E<B4>
$0005, 0005  KITPBUS_A<B5>
       .6    VSTUP1<V5>
       .7    VSTUP2<V5>
$0006, 0006  KITPBUS_B<B6>
$0007, 0007  KITPBUS_C<B7>
$0008, 0008  F<L8+0>
$0009, 0009  F<L8+1>
$000A, 0010  F<L8+2>
$000B, 0011  F<L8+3>
$000C, 0012  G<S12:20>
$000D, 0013  G[1]<B13>
$000E, 0014  G[2]<B14>
$000F, 0015  G[3]<B15>
$0010, 0016  G[4]<B16>
$0011, 0017  G[5]<B17>
$0012, 0018  G[6]<B18>
$0013, 0019  G[7]<B19>
$0014, 0020  G[8]<B20>
$0015, 0021  G[9]<B21>
$0016, 0022  G[10]<B22>
$0017, 0023  G[11]<B23>
$0018, 0024  G[12]<B24>
$0019, 0025  G[13]<B25>
$001A, 0026  G[14]<B26>
$001B, 0027  G[15]<B27>
$001C, 0028  G[16]<B28>
$001D, 0029  G[17]<B29>
$001E, 0030  G[18]<B30>
$001F, 0031  G[19]<B31>
...
Registry realnych cisel:
------------------------
$0000, 0000  C<R0>
```

6.7 Přetypování

Občas je třeba přetypovat symbolickou proměnnou jednoho typu na symbolickou proměnnou jiného typu. Je možno přetypovat pouze “vyšší” typ na “nižší” typ. Přetypování lze provést pouze v sekci symbol. Povoleno je následovné přetypování:

```
DateTime -> LongInt
LongInt  -> DateTime
LongInt  -> Byte
LongInt  -> Word
LongInt  -> Integer
Integer  -> Word
Integer  -> Byte
Word     -> Integer
Word     -> Byte
String   -> Byte
```

Samozřejmostí je přetypování na stejný typ.

Obecně jsou možné následující konstrukce:

```
Name1=Typ1(Name2)
```

Definuje novou symbolickou proměnnou Name1 typu Typ1, která začíná v poli registrů na stejném místě jako symbolická proměnná Name2.

Poznámka: je-li typ proměnné Name2 “vyšší” než typ proměnné Name1, leží proměnná Name1 na nižších bytech proměnné Name2.

```
Name1=Typ1(Name2[m])
Name1=Typ1(Name2+m)
```

Definuje novou symbolickou proměnnou Name1 typu Typ1, která začíná v poli registrů o m bytů výše než začíná symbolická proměnná Name2. Hodnota m musí být poplatná typu proměnné Name2, tj. pro Integer a Word musí být dělitelná 2, pro typ LongInt a DateTime dělitelná 4.

```
Name1=Typ1(Name2)[m]
Name1=Typ1(Name2)+m
```

Definuje novou symbolickou proměnnou Name1 typu Typ1, která začíná v poli registrů o m bytů výše než začíná symbolická proměnná Name2. V tomto případě však hodnota m musí být poplatná typu Typ1.

Jinými slovy lze obsah předchozích dvou odstavců napsat: přetypování se provádí podle uzávorkování s tím pravidlem, že při vyhodnocování musí každý stupeň vyhodnocení mít význam.

Příklad:

```
SYMBOL
 Vstup=LongInt;
 VstupA=Byte(Vstup[0]);
 VstupB=Byte(Vstup)[1];
 VstupC=Byte(Vstup)+2;
 VstupD=Byte(Vstup)[3];
```

Příklad ukazuje nadefinování našeho obrazu vstupní brány, která má čtyři porty typu Byte. V paměti chceme, aby byly porty umístěny za sebou, a mohli jsme tak s nimi pracovat někdy jako s typem LongInt, jindy jako s typem Byte. Můžeme to tedy provést jedním ze způsobů, jak ukazuje příklad.

6.8 Deklarace HW-objektů a SW-objektů

Sekce CONFIGURATION je určena jak již její název napovídá ke konfiguraci systému.

V sekci CONFIGURATION se zapisují definice tzv. HW-objektů a definice tzv. SW-objektů.

Deklarace HW-objektů zapínají a konfigurují parametry ovladačů, umožňujících obsluhu jednotlivých HW komponent, ze kterých se HW sestava skládá. Pro správnou funkčnost ovladačů je nutná fyzická přítomnost daného HW.

Deklarace SW-objektů popisují definice tzv. SW-objektů, které si můžeme představit jako pevné algoritmy, naprogramované tvůrci systému, které běží paralelně s algoritmy, naprogramovanými uživatelem v procedurách MAIN a FAST nebo jsou z nich volány speciálními funkcemi. Běh těchto pevných algoritmů může uživatel ovlivnit pouze tak, že jim zadá různé parametry - buď v okamžiku definice takového objektu v sekci CONFIGURATION, nebo v průběhu provádění uživatelských procedur zápisem nových hodnot do uživatelských registrů, na které jsou dané SW objekty vázány.

V následujících kapitolách jsou na příkladech vysvětleny jednotlivé deklarace.

Pozn. Objekty, uvedené v této kapitole, avšak nepopsané v následujících kapitolách, nejsou dosud implementovány.

6.9 Deklarace procedur (vlastní řídící algoritmus)

Kromě definování parametrů systémových procesů (SW-objektů a HW-objektů) může uživatel programovat vlastní řídící algoritmus a to hned několika způsoby. V první řadě musí napsat základní algoritmus, který se má stále cyklicky opakovat a to pomocí zápisu procedury MAIN.

V rámci tohoto algoritmu může uživatel volat různé další procedury, které je však potřeba definovat dříve než budou volány (ve smyslu pořadí v zápisu celého programu, tj. "výše").

Paralelně s prováděním hlavní procedury MAIN je pravidelně prováděna i předem definovaná procedura s názvem FAST (každých 10, 20 nebo 50 ms). Uživatel má možnost si definicí procedury s uvedeným jménem určit, co se má pravidelně v nastaveném časovém inetrvalu provádět. Pozor, u procedury FAST nesmí být rozšiřující kód příliš obsáhlý, jinak nemusí dojít k jeho zpracování.

6.10 Procedura MAIN

```
PROCEDURE MAIN;
begin
 příkaz;
 příkaz;
 ....
end;
```

Označuje **povinnou** uživatelskou proceduru **MAIN**, v které je zapsán hlavní cyklicky se opakující algoritmus.

Před započetím algoritmu, uvedeného v proceduře MAIN, se přečtou všechny vstupní signály, uvedené v množině MAIN=[] ve všech HW-objektech.

Po ukončení algoritmu, uvedeného v proceduře MAIN, se provede zápis na všechny výstupní signály, uvedené v množině MAIN=[] ve všech HW-objektech.

Jestliže má být procedura MAIN prázdná, musíme uvést alespoň zápis:

```
procedure MAIN; begin end;
```

6.11 Procedura FAST

```
PROCEDURE FAST;
begin
 příkaz;
 příkaz;
 ....
end;
```

Označuje nepovinnou uživatelskou proceduru **FAST**, v níž je zapsán algoritmus, který se má provádět každých 10, 20 nebo 50 ms.

V těle této procedury by neměly být používány vůbec operace s registry typu real, neměl by být používán cyklus ani příliš dlouhý algoritmus, aby celková délka provádění procedury FAST byla podstatně menší než nastavená doba jejího pravidelného volání (10, 20 nebo 50 ms). Doporučuje se použití jednoduchých operací (přiřazovací příkaz, příkaz if, stand. procedura inc,dec ap.).

Pro nastavení periody opakování volání procedury FAST použijeme option FastFreq (viz kap. 6.17.2). Standartně nastavenou periodu 10ms můžeme předefinovat na 20ms, resp. 50ms.

Před započetím algoritmu uvedeného v proceduře FAST, se přečtou všechny vstupní signály, uvedené v množině FAST=[] ve všech HW-objektech.

Po ukončení algoritmu uvedeného v proceduře FAST, se provede zápis na všechny výstupní signály, uvedené v množině FAST=[] ve všech HW-objektech.

Procedura INIT

```
PROCEDURE INIT;
begin
 příkaz;
 příkaz;
 ....
end;
```

Označuje nepovinnou uživatelskou proceduru **INIT**, v které je zapsán algoritmus, který se má provádět vždy po resetu.

Po resetu se vytvoří počáteční hodnota registrů takto:

1. Všechny registry se vynulují.
2. Registry použité v HW-objektech, resp. SW-objektech, se nainicializují na svou implicitní hodnotu, resp. hodnotu zadanou v konfigurační sekci při definici příslušného objektu.
3. Registry uchované v SW-objektu **SAVER** obnoví svoji hodnotu tak, jak ji měly před provedením resetu při minulém běhu téhož programu.
4. Provede se algoritmus procedury **INIT.**
5. Do procedury INIT tedy zpravidla zapisujeme pouze inicializaci všech uživatelských registrů, které mají mít konkrétní počáteční hodnotu různou od nuly, nepatří do žádného SW-objektu nebo HW-objektu a nemají mít hodnotu stejnou jako při posledním běhu programu.
6. Do procedury INIT však též můžeme zapsat prvotní hodnotu registrů, uchovávaných přes RESET objektem SAVER a to například takto:

```
PROCEDURE INIT;
begin
 ...
 if (I100<100) or (I100>1000) then I100:=500;
 ...
end;
```

Předpokládáme, že uživatelem nebo procesem reálně nastavená hodnota může být v rozmezí 100 až 1000, prvotní inicializační hodnota (po prvním spuštění čerstvě nahraného programu) má být 500.

Rozdíl mezi inicializací registru, přiřazeného SW, resp. HW-objektu v rámci definice tohoto objektu v **konfigurační** sekci a inicializací téhož registru v proceduře **INIT** je ten, že inicializace v rámci **konfigurační** sekce může být změněna nahráním hodnoty z SW-objektu **SAVER**. Například:

Příklad 1

```
configuration
 SWOBJ=TERM10,NAME=T,ADR=$2300,VAR=B0:3,
       LED=$AA;
```

```
 SWOBJ=SAVER, SAVE=[B0..B10];
procedure MAIN;
begin
 if ... then T_LED:=not T_LED;
end;
```

Příklad 2

```
configuration
 SWOBJ=TERM10, NAME=T, ADR=$2300, VAR=B0:3;
 SWOBJ=SAVER, SAVE=[B0..B10];
procedure INIT;
begin
 T_LED:=$AA;
end;
procedure MAIN;
begin
 if ... then T_LED:=T_LED+1;
end;
```

V prvním příkladu mají diody LED terminálu prvotní hodnotu $AA, avšak po dalším resetu už může být jiná, dle naposledy změněné hodnoty. Naproti tomu v druhém příkladu mají diody LED po resetu vždy hodnotu $AA.

6.12 Uživatelská procedura

```
PROCEDURE XXX;
begin
 příkaz;
 příkaz;
 ....
end;
```

Uživatel může nadefinovat libovolný počet uživatelských procedur. Uživatelské procedury je možno definovat pouze jednoúrovňově (tj. ne uvnitř jiných procedur). Na příkladu je uvedena uživatelská procedura XXX (libovolný identifikátor jinde nepoužitý), která popisuje dílčí algoritmus, který chce uživatel použít v rámci jiných procedur. Uživatelské procedury nemohou mít na rozdíl od standardních procedur parametry.

Uživatelskou proceduru voláme z jiné procedury příkazem volání procedury - kap.6.13.2

Počet vnořených volání uživatelských procedur není omezen.

6.13 Příkazy a operace

6.13.1 Přiřazovací příkaz

```
registr:=výraz;
```

resp.

```
registr=výraz;
```

Registru na levé straně lze přiřadit hodnotu výrazu na pravé straně. Při vyhodnocování výrazu se uvažuje priorita operátorů a závorky dle běžných pravidel, automaticky je prováděna případná konverze typů. Podrobně je vyhodnocování výrazů popsáno v kap. 6.14 .

Je-li hodnota výrazu mimo meze povolené pro typ registru na levé straně, přiřadí se do registru minimální, resp. maximální, přípustná hodnota pro daný typ registru. Například po provedení příkazů:

```
L100:=100000;
I104:=L100;
B106:=I104;
```

se do registru I104, který je typu integer, přiřadí hodnota 32767 a do registru B106, který je typu byte, hodnota 255.

Je-li hodnota typu real přiřazována do celočíselné proměnné, provede se zaokrouhlení. Například po provedení příkazů:

```
R1  :=1234.56;
I104:=R1;
B106:=R1;
```

se do registru I104, který je typu integer, přiřadí hodnota 1235 a do registru B106, který je typu byte, hodnota 255.

Vícenásobné přiřazení (možné v jazyku C) zde není definováno, proto zápis

```
A = B = 0;
```

způsobí vyhodnocení výrazu "B=0" a jeho výsledná hodnota (true=1 resp. false=0) je přiřazena do registru A. U tohoto a podobného zápisu je tedy vždy první symbol "=" chápán jako přiřazovací příkaz, (může být místo něho symbol ":="), další symboly "=" jsou chápány jako součást výrazu, jehož výsledek má být přiřazen.

6.13.2 Příkaz volání procedury

```
XXX;
```

Příkaz **volání uživatelské procedury**, případně standardní procedury bez parametrů.

```
XXX( seznam parametrů);
```

Příkaz **volání standardní procedury** s parametry (obsluha COM-kanálu, archívu, stringové operace ap.).

6.13.3 Složený příkaz

```
BEGIN příkaz1; příkaz2; ...END
```

Tento složený příkaz lze použít všude tam, kde můžeme použít **příkaz**. Způsobí postupné provedení všech příkazů mezi příkazovými závorkami **begin** a **end**.

6.13.4 Podmíněný příkaz

```
IF Výraz THEN příkaz1;
příkaz3;
```

Pokud je **výraz** pravdivý, tj. má hodnotu <>0, provede se **příkaz1**, pokud je nepravdivý, tj. má hodnotu 0, **příkaz1** se neprovede. Poté se provede další příkaz **příkaz3.** Způsob vyhodnocování výrazů je uveden podrobně v kap. 0 .

```
IF Výraz THEN příkaz1
         ELSE příkaz2;
příkaz3;
```

Pokud je **výraz** pravdivý, tj. má hodnotu <>0, provede se **příkaz1**, pokud je nepravdivý, tj. má hodnotu 0, provede se **příkaz2**. Poté se provede další příkaz **příkaz3.**

6.13.5 Příkaz větvení

```
CASE Výraz OF
 hodnota1 : příkaz1;
 hodnota2 : příkaz2;
 ...
 hodnotaN : příkazN;
 ELSE příkazE;
END;
```

Výraz je testován na rovnost s konstantními hodnotami **hodnota1** až **hodnotaN.** V případě shody se provede jeden z příslušných příkazů **příkaz1** až **příkazN**, v případě neúspěchu u všech testovaných hodnot se provede **příkazE** za klíčovým slovem **ELSE**. Není-li část s klíčovým slovem **ELSE** zapsána, neprovede se žádný příkaz.

Výraz může být název registru nebo i složitější výraz typu bit nebo byte. Způsob vyhodnocování výrazů je uveden podrobně v kap. 6.14.

Na místě **hodnotax** můžeme zapsat buď jednu konstantu ve tvaru

```
const
```

nebo více konstant oddělených čárkou ve tvaru

```
const1,const2,...,constk
```

nebo interval konstant ve tvaru

```
const1..const2
```

nebo několik intervalů konstant ve tvaru

```
const1..const2,const3..const4
```

Výše uvedené konstanty musejí být typu byte, resp. bit, ve shodě s typem výrazu **výraz.**

6.13.6 Příkaz cyklu REPEAT

```
REPEAT
 příkaz1;
 příkaz2;
 ...
 příkazN;
UNTIL výraz;
```

Způsobí opakované provádění příkazů **Příkaz1** až **PříkazN** až do doby, kdy **výraz** nabude pravdivé hodnoty, tj. hodnoty<>0.

V tomto typu cyklu se příkazy, uvedené v těle cyklu provedou nejméně 1x.

Aby tento příkaz cyklu nebyl nekonečný, musíme buď do těla cyklu vložit příkaz, který modifikuje nějakou proměnnou ve **výraz** a tím dojde k ukončení cyklu, nebo musí být nějaká proměnná ve **výraz** závislá na měnících se vnějších podmínkách (např. čtená hodnota z IO-vstupu). V druhém případě je potřeba zajistit, aby ke zmněně vnějších podmínek mohlo dojít (volání funkce WAIT).

V případě, že bychom naprogramovali nekonečný cyklus (a uvnitř cyklu nevolali systémovou proceduru WAIT), provede se po chvíli automaticky RESET celého programu.

6.13.7 Příkaz cyklu WHILE

```
WHILE výraz DO příkaz;
```

Způsobí vyhodnocení **výraz** a v případě jeho nenulové hodnoty způsobí (opakované) provádění příkazu **příkaz** (což může být např. složený příkaz **begin příkaz1; příkaz2; ... end**).

V tomto typu cyklu se příkazy, uvedené v těle cyklu nemusí provést ani 1x (jestliže je od počátku **výraz** nepravdivý, tj. nulový).

Aby tento příkaz cyklu nebyl nekonečný, musíme buď do těla cyklu vložit příkaz, který modifikuje nějakou proměnnou ve **výraz** nebo musí být nějaká proměnná ve **výraz** závislá na měnících se vnějších podmínkách (např. čtená hodnota z IO-vstupu).

V případě, že bychom naprogramovali nekonečný cyklus (a uvnitř cyklu nevolali systémovou proceduru WAIT), provede se po chvíli automaticky RESET celého programu.

6.13.8 Příkaz cyklu FOR

```
FOR Reg := výraz1 TO výraz2 DO příkaz;
```

resp.

```
FOR Reg = výraz1 TO výraz2 DO příkaz;
```

Příkaz cyklu s určeným konečným počtem kroků. **Příkaz** je vykonáván pro všechny hodnoty **Reg** počínaje hodnotou **výraz1** až po hodnotu **výraz2** včetně. Hodnota registru **Reg** se po provedení každého průchodu cyklem **zvětší o 1**. Reg nesmí být typu real.

V tomto typu cyklu se příkazy, uvedené v těle cyklu nemusejí provést ani 1x (jestliže je od počátku **výraz1** větší než **výraz2**).

```
FOR Reg := výraz1 DOWNTO výraz2 DO příkaz;
```

resp.

```
FOR Reg = výraz1 DOWNTO výraz2 DO příkaz;
```

Příkaz cyklu s určeným konečným počtem kroků. **Příkaz** je vykonáván pro všechny hodnoty **Reg** počínaje hodnotou **výraz1** až po hodnotu **výraz2** včetně. Hodnota registru **Reg** se po provedení každého průchodu cyklem **zmenší o 1**. Reg nesmí být typu real.

V tomto typu cyklu se příkazy uvedené v těle cyklu nemusí provést ani 1x (jestliže je od počátku **výraz1** menší než **výraz2**).

6.13.9 Příkaz ukončení procedury

```
EXIT;
```

Tento příkaz způsobuje ukončení činnosti procedury (skok za poslední příkaz procedury). Použijeme jej pro ošetření chybových stavů, výjimek apod.

6.13.10 Příkaz MainExit

```
MAINEXIT;
```

Příkaz vyvolá skok na konec procedury Main .

6.13.11 Příkaz WAIT

```
WAIT;
```

Příkaz vyvolá přerušení zpracování programu MAIN, zavolání obsluhy systému (součástí je i obsluha vstupů a výstupů) a při následujícím zpracování MAINu se pokračuje další instrukcí.

6.13.12 Příkaz obecného skoku

```
GOTO návěští;
```

Po příkazu skoku se začnou vykonávat příkazy, které následují za deklarací **`návěští`**, která musí být uvedena v témže bloku, blokem zde myslíme uživatelskou proceduru, systémovou proceduru, nebo proceduru vytvořenou z popisu sekce onkey.

Deklarace návěští má tvar

```
navěští:
```

a lze jej umístit před libovolný příkaz, umístěný v těle procedury, ve které má být použit příkaz skoku **`goto.`** Jako návěští může být použit libovolný identifikátor, dosud nepoužitý. V příkazu skoku smí být použito návěští, které ještě nebylo definováno. Tato jediná výjimka slouží k tomu, aby bylo možné provést v kódu skok "dopředu".

Pozn. Příkaz skoku by měl uživatel použít jen ve zcela výjimečném případě, neboť jeho běžné použití vede k zneprůhlednění algoritmu. Raději je třeba používat příkazy cyklu aj.

6.13.13 Aritmetické operace

```
f:= reg1 + reg2;
f:= reg1 - reg2;
f:= reg1 * reg2;
f:= reg1 / reg2;
```

6.13.14 Logické operace

```
f:=NOT(reg);
f:=OR(reg);
f:=XOR(reg);
```

6.13.15 Operace relací

```
>,<,>=,<=,<>,=
```

6.13.16 Operace adresa

```
f:=addr(reg);
```

Operand addr vrátí adresu dané proměnné. Pro celečíselné proměnné vrátí číslo Bytu, pro proměnné typu real číslo daného real registru. Číslo se vrací vždy bez typu.

např.:

```
adresa:=addr(B0);   vrátí 0
adresa:=addr(L44);  vrátí 44
adresa:=addr(R7);   vrátí 7
```

6.13.17 Procedury inkrementace a dekrementace

```
INC(reg);
DEC(reg);
```

6.13.18 Operace posunů

```
f:=reg1 SHL reg2;
f:=reg1 SHR reg2;
```

6.13.19 Goniometrické funkce

```
f:=SIN(reg);
f:=COS(reg);
```

Goniometrické funkce mají dva režimy výpočtu. Pokud se jako parametr reg použije adresa celočíselného registru nebo konstanty, počítá se tzv. DEC formát vstupního úhlu – zadání ve °. Nejmenším krokem je 1°. Výstupem funkce je pak celočíselná hodnota v intervalu <–100,100> s krokem 1 odpovídající obvyklému oboru hodnot <-1,1>. Pokud je parametr reg adresou rálného registru, počítá se v tzv. RAD formátu vstupníhu úhlu – zadání v radiánech. Výstup funkce je pak v obvyklém oboru hodnot, v intervalu <-1,1>.

Příklady: DGONIO, DGONIOR

6.13.20 Další matematické funkce

```
f:=EXP(reg);
f:=LOG(reg);
f:=RND(reg);
```

RND – generátor celočíselných náhodných čísel z intervalu <0,Nb), kde Nb je parametr funkce RND.

Příklad: DRND

```
f:=SQR(reg);
f:=SQRT(reg);
```

6.13.21 Funkce výpočtu kódů CRC

```
f:=CRC8(reg1, reg2);
f:=CRC16(reg1, reg2);
```

funkce vypočtou CRC kód od adresy zadané registrem reg1 do adresy zadané reg2 včetně.

Příklad: DCRC

6.13.22 Procedury pro práci s typem DateTime (časem)

```
SETTIME(WTime);
PACKTIME(WTime, DTime);
UNPACKTIME(DTime, WTime);
```

kde Wtime je adresa ukazující do paměti na první položku časového údaje - rok. V paměti je předpokládáno rozmístění dalších parametrů patrné z tabulky:

číslo bytu	číslo regist-ru v příkladu	symbolický název registru v příkladu	Význam
Base+0	B50	YEAR	kalendářní rok 0-65535
Base+1	B51		pokračování CL_YEAR
Base+2	B52	MONTH	kalendářní měsíc 1-12
Base+3	B53	DAY	kalendářní den 1-31
Base+4	B54	DOW	den v týdnu 0=ne až 6=so
Base+5	B55	HOUR	hodiny 0-59
Base+6	B56	MIN	minuty 0-59
Base+7	B57	SEC	sekundy 0-59

kde Dtime je speciální formát pro uložení zapakovaného datumu a času s krokem 2 sec. Význam jednotlivých Byte je patrný z tabulky:

číslo bytu	číslo regist-ru v příkladu	symbolický název registru v příkladu	Význam
Base+0	B60	Dtime	komprimovaný údaj o datumu a času
Base+1	B61		pokračování DTime
Base+2	B62		pokračování DTime
Base+3	B63		pokračování DTime

Pomocí standardních procedur PACKTIME a UNPACKTIME můžeme jednotlivé formáty převádět mezi sebou s tím, že jako první parametr uvádíme zdroj, jako druhý parametr cíl operace:

Pomocí standardní procedury SETTIME(reg) můžeme nastavit hodiny reálného času. Registr reg musí být typu word a musí být počátkem 8-bytového časového údaje v unpack-time formátu (význam jednotlivých registrů dle první tabulky).

Příklady: DHODINY, DHODINYG

6.13.23 Procedura pro práci se stringy

```
ADDS(reg1,reg2);
```

Procedura připojí na konec stringového registru reg1 obsah stringového registru reg2. Při připojení se progádí kontrola maximální délky registru reg1.

6.13.24 Procedury přesunu – Move, MoveXXX

```
MOVE(reg1,reg2,reg3);
```

procedura přesune z adresy reg1 na adresu reg2 počet byte zadanych v reg3.

```
MOVEREAL(reg1,reg2);
```

procedura přesune z adresy reg1 na adresu reg2 6 bytu, - používá se pro přesun dat typu REAL do oblasti celočíselných registrů a naopak. Odpovídá uložení čísla Real v 6ti celočíselných registrech. Jeden operand musí být registr typu REAL, druhý celočíselný registr.

```
MOVESINGLE(reg1,reg2);
```

procedura převede hodnotu real čísla uloženého v registru typu REAL do 4-Bytové reprezentace real čísla a přesune ho do oblasti celočíselných registrů a naopak. Převod se uskutečňuje vždy z adresy reg1 na adresu reg2. Jeden operand musí být registr typu REAL, druhý celočíselný registr.

```
MOVEDOUBLE(reg1,reg2);
```

procedura převede hodnotu real čísla uloženého v registru typu REAL do 8-Bytové reprezentace real čísla a přesune ho do oblasti celočíselných registrů a naopak. Převod se uskutečňuje vždy z adresy reg1 na adresu reg2. Jeden operand musí být registr typu REAL, druhý celočíselný registr.

Pozn – tato instrukce není zatím implementována.

6.13.25 Procedury PROCi

```
PROCi(reg1,reg2,reg3);
```

procedury PROCi jsou vývojové procedury sloužící pro rychlou realizaci zákaznických požadavků. K dispozici je 10 procedur PROC0 až PROC9, s parametry reg1 až reg3. Přesný popis použitých procedur najdete v následujících kapitolách, nebo je dodáván se zákaznickou verzí programového vybavení.

6.13.26 Procedura obsluhy tiskárny

```
PROC0(Data,Adresa,reg);
```

Pro jednoduchý tisk na tiskárnu (port LPT) je určena universální procedura **PROC0**. Parametr **Data** je typu string a obsahuje jednu řádku textu (připravíme např. pomocí objektu GenStr kapitola 7.3), která se má na tiskárnu vytisknout. Procedura sama zajistí odřádkování. Parametr **Adresa** je adresa portu LPT, ke kterému je tiskárna připojena. Adresu získáte z dokumentace k dodaným komponentám. Parametr **reg** je libovolný registr, který je u této funkce pouze z důvodu kompatibility. Momentálně je procedura realizována tak, že nevrací uživateli potvrzení, zda k tisku došlo či nikoli. Tato skutečnost je dána odlišnou komunikací s různými tiskárnami. Bez patřičného ovladače pak nejsme tuto funkci schopni univerzálně zajistit. V případě potřeby je možno po individuální konzultaci danou proceduru o tuto funkci rozšířit.

Příklad: DLPT

6.13.27 Procedury obsluhy displeje řady LED

```
PROC1(Conf,reg,reg);
PROC2(Data,reg,reg);
```

Pro jednoduchý výpis textu na displej řady LED jsou určeny universální procedury **PROC1** a **PROC2**. Procedurou **PROC1** se provede inicializace displeje LED, procedurou **PROC2** vlastní přepis textu na displej. Parametr **Conf** je typu string a obsahuje konfigurační string displeje, parametr **Data** je také typu string a obsahuje text (připravíme např. pomocí objektu GenStr kapitola 7.3), který se má na displej zobrazit.

Struktura inicializačního stringu je následující: "ADR=$d221 OUT=$d225 BDI=6 BCL=5 BLO=4 JAS=15"

parametr	*Význam*
ADR	Adresa portu LED – zasílání dat
OUT	Adresa portu LED – aktivace datového portu
BDI	Číslo bitu signálu BDI
BCL	Číslo bitu signálu BCL
BLO	Číslo bitu signálu BLO
LAS	Hodnota jasu [0-15]

Adresy i čísla bitů získáte z dokumentace k dodaným komponentám. Parametr **reg** je libovolný registr, který je u procedur použit pouze z důvodu kompatibility.

Příklad: DLEDKIT

6.14 Vyhodnocování výrazů

V přiřazovacím příkazu, podmíněném příkazu, příkazu větvení a všech příkazech cyklu se vyskytuje pojem **`výraz`**.

Výrazem rozumíme matematický zápis, skládající se z jednotlivých operandů a operátorů. Výsledná hodnota a typ výrazu je dána prioritou operátorů a hodnotou jednotlivých operandů.

Priorita operátorů při vyhodnocování výrazů je následující:

1. nejprve se provádějí operace unární "+" resp."**-**", logická operace **not** , výpočet části výrazu uzavřeného do závorek a vyhodnocení standardních funkcí
2. poté se provádějí operace "*", "/", **and**, **shl**, **shr**, **div** a **mod**
3. poté se provádějí operace "+", "-", **or**, **xor**
4. nakonec se provádějí relační operace **> , < , >= , <= , <> , =**

Patří-li dvě operace do stejné prioritní skupiny, vyhodnocují se ve výrazu zleva doprava.

Každá z výše uvedených operací má definovaný typ výsledku v závislosti na typu jednoho, resp. dvou operandů, se kterými pracuje. Typ výsledku pro jednotlivé operace je přehledně uveden v následující tabulce:

operace	typ 1. operandu	typ 2. operandu	typ výsledku
Not	bit		bit
	byte		byte
	word		word
	jiný typ		**undef**
unární +, -	byte		integer
	word		integer
	integer		integer
	longint		longint
	real		real
"*", "/", "+", "-"	byte	byte	byte
	byte	word	word
	byte	integer	integer
	byte	longint	longint
	byte	real	real
	word	byte	word
	word	word	word
	word	integer	integer
	word	longint	longint
	word	real	real
	integer	byte	integer
	integer	word	integer
	integer	integer	integer
	integer	longint	longint
	integer	real	real
	longint	byte	longint
	longint	word	longint
	longint	integer	longint
	longint	longint	longint
	longint	real	real
	real	lib.čísel.typ	real
	ostatní	kombinace	**undef**
xor, or, and	bit	bit	bit
	byte	byte	byte
	byte	word	word
	word	byte	word
	word	word	word
	ostatní	kombinace	**undef**
shl, shr	byte	byte	byte
	word	byte	word
	ostatní	kombinace	**undef**
div, mod	byte	byte	byte

	byte	word	word
	byte	integer	integer
	byte	longint	longint
	word	byte	word
	word	word	word
	word	integer	integer
	word	longint	longint
	integer	byte	integer
	integer	word	integer
	integer	integer	integer
	integer	longint	longint
	longint	lib.čísel.typ	longint
	ostatní	kombinace	**undef**
<, >, <=, >=, <>, =	lib.čísel.typ	lib.čísel.typ	bit

stand. funkce	typ 1.operandu	typ 2.operandu	typ výsledku
Sizeof	libovol.typ	-	byte
Length	string	-	byte
Abs	integer	-	word
Abs	longint	-	longint
Sgn	lib.čísel.typ	-	integer
Min, Max	byte	lib.čísel.typ	byte
Min, Max	word	lib.čísel.typ	word
Min, Max	integer	lib.čísel.typ	integer
Min, Max	longint	lib.čísel.typ	longint
Min, Max	real	lib.čísel.typ	real
sin, cos, ln, exp, sqrt	lib.čísel.typ	-	real
Rnd	lib. celočís. typ	-	lib. celočís. typ
Adds	string	string	string

Pozn.: Operace div a mod nebyly dosud implementovány.

Výše uvedená pravidla je podstatné znát zejména při vyhodnocení složitějších výrazů. V následující tabulce jsou uvedeny některé příklady výrazů s uvedením typu a hodnoty výsledku. Pro jednoznačné určení typu operandů použijeme nesymbolických jmen registrů. Ve výrazech však samozřejmě můžeme používat i symbolická jména registrů.

Ve výrazech jsou uvedeny i konstanty. Typ konstanty je dán její velikostí a tím, jestli obsahuje unární +, resp. -.

Typ výsledku operace je velmi důležitý, protože určuje, zda výsledek operace může být spočten bezchybně nebo zda je nahrazen maximální, resp. minimální, možnou hodnotou pro daný typ výsledku (viz kapitola 6.15).

Typickým důsledkem je například neplatnost komutativního zákona u výrazu B1+B2-B3, pokud při některé z dílčích operací dojde k přetečení.

Obecně lze říci, že pokud uživatel zvolí dostatečně "velký" typ svých proměnných (například real nebo longint), nemusí se obávat zde uvedených nesnází a výsledek mu vyjde "matematicky" dobře. Jestliže chce však uživatel "šetřit" datový prostor i čas provádění instrukcí, musí vždy při zápisu výrazu zvážit, zda nebude spočten jinak, než si původně představoval.

Nechť

```
B10=1,B12=100,B14=220,
W20=1,W22=100,W24=255,W26=60000,W28=$ff00,
I30=1,I32=100,I34=220,I36=30000,I38=-1,
L40=1,L44=-100000,L48=-0,
R1=1.0,R2=100.0,R4=255.0,R5=5.5,R6=0,
R7=100.0,R8=1.23E23,
B0.0=0,B0.1=1
```

Pak vyhodnocení následujících výrazů vypadá takto:

výraz	výsledná hodnota	typ výsledku
a) konstanty, konstantní výrazy		
50	50	byte
500	500	word
+500	500	integer
-500	-500	integer
500000	500000	longint

5.0	5.0	real
2*(2-2*(4+2*(130+120)))	-2012	integer
2*2-2*4+2*130+120	376	integer
b) priority operátorů, operace s více typy operandů		
I30+I32*I34	22001	integer
I30+I32*B14	22001	integer
R1+R2*R4	25501.0	real
R8+R7	1.23E23	real
R1+I34/I32	3	real
R1+R4/I32	3.55	real
W20 and not W28	1	word
-B10+B12	99	integer
-B14+B10+B12	-119	integer
B10+B12*(+B14)	22001	integer
B10+B12*(B14+L48)	22001	longint
c) výrazy vedoucí k přetečení		
B10+B12*B14	255	byte
I30+B12*B14	256	integer
B12-B14+B10	1	byte
d) logické výrazy		
(B10=W20) and (B12=I32)	1	bit
B0.0 or B0.1 xor B0.1	0	bit
B0.1 and (I32>L44)	1	bit
R1=B10	1	bit
e) použití standardních funkcí		
size-of(B10)+sizeof(W20)+sizeof(L30)+sizeof(L40)	9	byte
sign(-450)	-1	integer
sign(450)	1	integer
abs(I38)	1	word

6.15 Ošetření chyb při vyhodnocování výrazů

Při vyhodnocování výrazů dochází při zpracování kódu k automatickým kontrolám rozsahů jednotlivých operandů a výsledku. Základní pravidlo zní: Pokud je přiřazovaná hodnota mimo rozsah cílového registru, dochází k jejímu oříznutí na povolenou mez danou rozsahem cílového registru. V praxi to znamená, že pokud se snažíme např. do registru typu BYTE umístit hodnotu 300 (rozsah byte je 0 až 255), dojde k nahlášení chyby přetečení a do registru je umístěna hodnota 255.

Vedle tohoto způsobu ošetření možného vzniku chyby je speciálně ošetřena operace dělení (dělení 0, 0 dělena 0), funkce ln, log atd. .

6.16 Systémové definice pro lexikální analyzátor

6.16.1 definice #define

Touto definicí můžeme předefinovat libovolný lexikální element jazyka. Možno použít např. pro zkrácení klíčových slov. Uvedeme-li například v programu definici

```
#define pos position
```

překladač nahradí před překladem všechna slova **pos** slovem **position**.

6.16.2 definice #include

Nalezne-li při čtení zdrojového souboru lexikální analyzátor překladače definici

```
#include názevsouboru
```

nahradí tuto definici obsahem souboru s uvedeným názvem. Neobsahuje-li název souboru příponu, uvažuje se přípona **.PRI**. Neobsahuje-li název souboru cestu (path), uvažuje se cesta, uvedená případně při zadávání jména zdrojového programu, jinak se uvažuje aktuální adresář.

6.17 Systémové definice - options

Ve zdrojovém programu může být uvedena speciální sekce **options**, ve které je možno nadefinovat některé systémové parametry pro překladač nebo procesor.

6.17.1 Definice ProgVer

Uvedeme-li ve zdrojovém programu definici

```
options
 ProgVer=xx;
```

kde xx je číslo v rozsahu byte (0..255), zadáme tím číslo verze uživatelského programu. Jestliže tuto definici neuvedeme, uvažuje se verze 255. Tato verze je uvedena spolu s názvem programu (stejným jako název souboru s hlavním programem) v hlavičce binárního souboru *.BIN. Verzi programu můžeme převést do uživatelské proměnné typu byte použitím systémové procedury SysProgVer - viz. kap. a zobrazit jí například informativně na některé obrazovce. Tato verze programu se též automaticky uvádí v Error-archívu (viz kap. 10.4, str. 51), aby bylo možno zjistit, jaká verze uživatelského programu vyvolala příslušnou chybu.

6.17.2 Definice FastFreq

Uvedeme-li ve zdrojovém programu definici

```
options
 FastFreq=20;
```

(možné další hodnoty jsou 10 a 50), změní se základní frekvence pro vyvolávání procedury FAST z implicitní hodnoty 10 ms na 20ms, resp. 50 ms.

Toto option použijeme tehdy, jestliže potřebujeme natolik složitou obsluhu v proceduře FAST, že se již nestíhá provádění této procedury každých 10ms. (To, že došlo k "přetečení" času při použití procedury FAST, poznáme přečtením systémového registru SysFastOverFlow - viz kap.11).

7. Terminály a tvorba obrazovek

7.1 Úvod

Jednou ze součástí aplikace může být potřeba prohlížet, případně měnit obsahy jednotlivých proměnných používaných v dané aplikaci. K tomuto účelu slouží v prostředí KIT-BUILDER několik speciálních objektů. V této kapitole si uvedeme přehled těchto objektů, popíšeme jejich konfigurační řádku a doplníme popisem jednotlivých formátovacích, textových, grafických a ovládacích prvků, ze kterých se jednotlivé obrazovky dají sestavovat. V poslední kapitole si uvedeme, které prvky lze na daných terminálech používat.

7.2 Terminál řady TERM10

```
CONFIGURATION
 HWOBJ=TERM10, NAME=T10, ADR=$2300;
```

Objekt popisuje operátorský panel - terminál TERM10.

V konfigurační části se pouze deklaruje, že takový terminál existuje, uvádí se jeho jméno (využívá se jako prefix pro sestavení symbolických adres řídících registrů – v tomto příkladě T10), a HW adresa ADR, na které je TERM10 připojen do sestavy. Parametr NAME je nepovinný, pak se k objektu přistupuje pod jménem TERM10.

Při překladu jsou danému objektu z pole uživatelských registrů automaticky přiděleny první 3 volné registry jejichž význam je uveden v následující tabulce:

číslo bytu	*Číslo registru v příkladu*	*symbolický název registru v příkladu*	*Význam*
Base+0	B1000	T10_SCRNO	číslo aktivní obrazovky
Base+1	B1001	T10_LED	signalizace pomocí diod LED
Base+2	B1002	T10_CTRL	řídící byte, obsahuje následující bitové příznaky
Base+2	B1002.0	T10_BEEP	zvuková signalizace
Base+2	B1002.1	T10_START	1=Bylo stisknuto tlačítko Start
Base+2	B1002.2	T10_STOP	1=Bylo stisknuto tlačítko Stop

Pokud nechceme využít automatické přidělení registrů, můžeme registry umístit ručně. To provedeme zadáním parametru **VAR=B1000:3**, který říká, že HW objekt TERM10 používá uživatelské registry B1000 až B1002, tj. celková délka vyhrazeného prostoru činí 3 byty. Údaj **:3** je při definici nepovinný, doporučuje se však jej uvádět, aby měl uživatel přehled o počtu bytů, které tento HW objekt mapuje.

Podle registru T10_SCRNO se určuje, která definice obrazovky (zápis TERMINAL) je momentálně aktivní. Změnou hodnoty tohoto registru se přechází na jinou obrazovku. Na počátku programu je třeba dbát na to, aby existovala definice obrazovky (např. č.0), která je implicitně požadována (každý registr má implicitně na počátku hodnotu 0).

Registr T10_LED se promítá na přední panel terminálu tak, že 0-tý bit je umístěn nejvíce vlevo, hodnota 0 nesvítí, 1 svítí.

Je-li registr T10_Beep nenulový, vydává se zvuková návěšť.

Registry T10_START, resp. T10_STOP, jsou v případě stisknutí patřičného tlačítka naplněny hodnotou 1. Pokud chceme tlačítko ošetřit, je třeba obsah registrů cyklicky testovat. V případě detekce stisknutého tlačítka musíme pro další detekci příslušný registr vynulovat!

Popis jednotlivých obrazovkek a reakce na stisk jednotlivých kláves (vyjma kláves Start a Stop) je uveden v deklaracích TERMINAL - viz. kapitola 7.5.

Příklady: DTEXT, DGRAF

7.3 Terminál TERM01

```
CONFIGURATION
 HWOBJ=TERM01, NAME=T01, ADR=$2320, IRQ=3;
```

Objekt popisuje operátorský panel - terminál TERM01.

V konfigurační části se pouze deklaruje, že takový terminál existuje, uvádí se jeho jméno (využívá se jako prefix pro sestavení symbolických adres řídících registrů – v tomto příkladě T01), HW adresa ADR a číslo přerušení IRQ komunikační linky, prostřednictvím které je terminál připojen k procesorové desce. Parametr NAME je nepovinný, pak se k objektu přistupuje pod jménem TERM01.

Při překladu je danému objektu z pole uživatelských registrů automaticky přidělen první volný registr jehož význam je uveden v následující tabulce:

číslo bytu	*Číslo registru*	*symbolický název registru*	*Význam*

	v příkladu	*v příkladu*	
Base+0	B1000	T01_SCRNO	číslo aktivní obrazovky

Pokud nechceme využít automatické přidělení registrů, můžeme registry umístit ručně. To provedeme zadáním parametru **VAR=B1000:1**, který říká, že HW objekt TERM01 používá uživatelský registr B1000, tj. celková délka vyhrazeného prostoru činí 1 byte. Údaj **:1** je při definici nepovinný, doporučuje se však jej uvádět, aby měl uživatel přehled o počtu bytů, které tento HW objekt mapuje.

Podle registru T01_SCRNO se určuje, která definice obrazovky (zápis TERMINAL) je momentálně aktivní. Změnou hodnoty tohoto registru se přechází na jinou obrazovku. Na počátku programu je třeba dbát na to, aby existovala definice obrazovky (např. č.0), která je implicitně požadována (každý registr má implicitně na počátku hodnotu 0).

Popis jednotlivých obrazovkek a reakce na stisk jednotlivých kláves je uveden v deklaracích TERMINAL - viz. kapitola 7.5.

Příklad: DTERM01

7.4 Generátor stringů GENSTR

```
CONFIGURATION
 SWOBJ=GENSTR, NAME=STR, BUFF=S10:30;
```

Objekt popisuje obecný generátor stringů resp. výstupních zpráv, definovaných nad výstupním buffrem. K objektu se chováme jako k obecné výstupní konzoli, na kterou můžeme zapisovat textové zprávy pomocí formátu, sestaveného pomocí popisů, uvedených v kap. 7.5.

V sekci **configuration** se pouze deklaruje existence tohoto generátoru, adresa výstupního bufferu a jméno generátoru.

Při překladu je danému objektu z pole uživatelských registrů automaticky přidělen první volné registr jehož význam je uveden v následující tabulce:

Číslo bytu	*číslo registru v příkladu*	*symbolický název registru v příkladu*	*Význam*
Base+0	B1000	STR_SCRNO	číslo aktivní zprávy

Pokud nechceme využít automatické přidělení registrů, můžeme registry umístit ručně. To provedeme zadáním parametru **VAR=B1000:1**, který říká, že SW objekt GENSTR používá uživatelský registr B1000, tj. celková délka vyhrazeného prostoru pro řídící proměnné generátoru činí 1 byte. Údaj **:1** je při definici nepovinný, doporučuje se však jej uvádět, aby měl uživatel přehled o počtu bytů, které tento SW objekt mapuje.

Podle registru STR_SCRNO se určuje, která definice formátu (formátovací popis TERMINAL) je momentálně aktivní. Změnou hodnoty tohoto registru se přechází na jiný formát výpisu. Neaktivní formát, který nesmí být u tohoto objektu nikdy nadefinován, má číslo 0.

V klidu má proměnná STR_SCRNO hodnotu 0, formát je neaktivní, žádný text se negeneruje. V okamžiku, kdy je provedena změna proměnné STR_SCRNO, vybere se příslušný formátovací popis TERMINAL, provede se vygenerování textu dle vybraného formátu do výstupního bufferu a proměnná STR_SCRNO opět obdrží hodnotu 0.

Parametr **BUFF=S10:30** říká, že SW objekt GENSTR používá pro výstupní buffer uživatelský registr S10 o délce 30 znaků.

Formátovací popis TERMINAL pro objekt GENSTR smí obsahovat pouze vybrané popisy PRINT a CASE - viz kap. 7.6.

7.5 Prvky na obrazovce

7.5.1 Úvod

V této kapitole je vysvětlen způsob definování výpisů na obrazovku terminálu. Úvodem je třeba zdůraznit, že proces obsluhující obrazovku terminálu běží paralelně s hlavní procedurou MAIN a že algoritmy, které se mají provést jako reakce na stisk jednotlivých kláves na terminálu (vyjma režimu editace) se provádějí vždy po skončení jednoho průběhu procedurou MAIN.

Všechny ukázkové příklady jsou většinou psány pro terminál TERM10. V kapitole 7.6 je uvedena možnost použití jednotlivých příkazů pro různé jiné typy terminálů.

Každá obrazovka se popisuje zvláštní deklarací TERMINAL. Typická deklarace TERMINAL vypadá např. takto:

```
terminal T10:0;                                          { 1}
begin                                                    { 2}
 bitmap 0; font 2;                                       { 3}
 position 10,113; print "R10=", R10:10:3;                { 4}
 rect 9,112,109,122;                                     { 5}
 onkey                                                   { 6}
```

```
  '1':R10:=1;                                          { 7}
  '2':R10:=2;                                          { 8}
 end;                                                  { 9}
help                                                   {10}
 font 1; position 1,1;print "Toto je help text";       {11}
end;                                                   {12}
```

Na řádce {1} je uvedena hlavička deklarace TERMINAL. Obsahuje název terminálu, který musí být shodný se jménem terminálu, uvedeným v SW-objektu terminálu, např. v SW-objektu TERM10 (viz kap.7.2). Pokud nebyl při definici SW-objektu TERM10 použit parametr NAME, je implicitním jménem TERM10. Jako druhý parametr hlavičky je třeba uvést číslo obrazovky.

Pokud v INIT proceduře nebo přímo v SW-objektu TERM10 nenadefinujeme jinak, má po resetu registr TERM10_ScrNo hodnotu 0, a proto se jako první zobrazí obrazovka č.0 - tuto obrazovku proto v tomto příkladu musíme deklarovat.

Deklarace TERMINAL popisuje

1. co se má vypisovat na obrazovku
2. co se má provést jako reakce na stisk jednotlivých kláves.

Každá obrazovka terminálu se skládá ze tří vrstev:

1. Vrstva pPodkladové bitmapy
2. Vrstva pro výpis textů
3. Vrstav pro vykreslení vektorové grafiky

V následujících kapitolách jsou popsány jednotlivé popisy, definující jednotlivé zobrazované elementy na obrazovce terminálu.

Pro definici první vrstvy (podkladová bitmapa) je určen popis BITMAP.

Pro definici druhé vrstvy (textové výpisy) jsou určeny popisy, uvedené v kapitolách 7.5.3 až 7.5.7.

Pro definici třetí vrstvy (výkreslení grafiky) jsou určeny popisy, uvedené v kapitolách 7.5.9 až 7.5.16.

Pro popis reakcí na stisk klávesy slouží popisy EDIT (kap.7.5.6) včetně popisu EDITENTER (kap. 7.5.8), dále popisy EDITP (kap. 7.5.7) a ONKEY(kap.7.5.17).

7.5.2 Popis BITMAP

První vrstvu (podkladovou bitmapu) definujeme popisem **`bitmap x.`** , uvedeným na řádku {3}. číslo 0 je vyhrazeno pro implicitní podkladovou bitmapu s logem SofCon, následující čísla jsou určena pro uživatelem definované bitmapy. Tyto uživatelské bitmapy můžeme vytvořit např. v programu Paintbrush, který je součástí o.s.MS-Windows (v černobílém módu) a před spuštěním programu nahrát do terminálu.

Pozor, při použití podkladové bitmapy jsou všechny textové a grafické prvky definované před uvedením příkazu BITMAP smazány.

Příklad: DGRAF

7.5.3 Popis FONT

Druhou vrstvu - výpis textu definujeme tak, že nejprve určíme typ fontu, kterým se má vypisovat (viz řádka {3}, popis **`font`**). Uvedeme-li číslo fontu 0 nebo 1, vypisuje se text implicitně danými fonty 0 nebo 1 - viz demo příklad D_TEXT. Uvedeme-li číslo fontu 2 až cca 19, provádí se výpis dle uživatelem nahraného fontu.

Typ fontu platí pro všechny následující popisy pro druhou vrstvu, tj. **`PRINT`** a **`EDIT.`**

Příklad: DTEXT

7.5.4 Popis POSITION

Pro popis druhé vrstvy **`PRINT`** resp. **`EDIT`** (uvedené dále), musíme určit pozici na obrazovce (horní levý bod prvního vypisovaného znaku), od které se má popis umístit. (viz řádka {4}, popis **`position`**).

Souřadnice na obrazovce se měří zásadně od levého horního rohu, který má souřadnice 0,0. U terminálu TERM10 jsou souřadnice pravého dolního rohu 239,127.

Pokud není pozice explicitně uvedena, je na počátku sekce **`TERMINAL`** standardně nastavena pozice 0,0, po uvedení jednoho z popisů **`PRINT`** nebo **`EDIT`** pak do "řádky" těsně za text, vygenerovaný předchozím popisem.

Nechceme-li používat dlouhý identifikátor **`position`**, můžeme jej zkrátit použitím systémové definice **`#define`** - viz kap. 6.16.1.

Příklad: DTEXT

7.5.5 Popis PRINT

Popisem **print** definujeme, jaký text má být vypsán v druhé vrstvě. (viz řádka {4}). V popisu **print** můžeme uvést tyto prvky:

1. Text, uzavřený do dvojitých uvozovek - v tomto případě se bude vypisovat uvedený text, tj. stále totéž.
2. Název celočíselného registru (dále označený jako "Reg"), - v tomto případě bude vypisovaný údaj proměnný tak, jak se bude měnit hodnota registru. RegB značí v dále uvedené tabulce celočíselný registr typu byte, RegW celočíselný registr typu word a RegT celočíselný registr typu DateTime.
3. Název reálného registru (dále označený jako "RegR").

Hodnota registru Reg se implicitně vypisuje dekadicky na nejmenší možný počet míst, můžeme však použít doplňující formátovací informace, uvedené v následující tabulce.

U registru RegR musíme vždy uvést doplňující formátovací informace, uvedené v tabulce,jinak se hlásí chyba při překladu.

argument popisu PRINT	význam
Reg	celočíselná hodnota se vypisuje dekadicky na nejmenší počet možných znaků
Reg:n	celočíselná hodnota se vypisuje dekadicky, doplňuje se zleva na celkem n znaků mezerami
Reg:n:d	celočíselná hodnota se vypisuje dekadicky, doplňuje se zleva na celkem n znaků mezerami, přičemž d znaků z konce se oddělí desetinnou tečkou
DEC Reg	dtto jako Reg
HEX Reg	hodnota se vypisuje hexadecimálně (byte na 2 znaky, word na 4 znaky, longint na 8 znaků)
BIN Reg	hodnota se vypisuje binárně ve tvaru xxxx_xxxx_xxxx_xxxx pro word a xxxx_xxxx pro byte.
TIME RegB	Pokud máme čas v registrech reprezentovaný v unpack formátu ve tvaru odpovídajícím tabulce (viz. kapitola 6.13.22) můžeme provádět jeho formátovaný tisk uvedením adresy registru obsahujicím údaj „hodiny“. Adresa musí být typu Byte. Časový údaj se pak vypíše ve tvaru hh:mm:ss
TIME RegD	Vypíše časový údaj odpovídající hodnotě registru typu DateTime. Výpis se provede ve tvaru hh:mm:ss
DATE RegW	Pokud máme datum v registrech reprezentovaný v unpack formátu ve tvaru odpovídajícím tabulce (viz. kapitola 6.13.22) můžeme provádět jeho formátovaný tisk uvedením adresy registru obsahujicím údaj „rok“. Adresa musí být typu Word. Datum se pak vypíše ve tvaru dd:mm:yyyy
DATE RegD	Vypíše datum odpovídající hodnotě registru typu DateTime. Výpis se provede ve tvaru dd:mm:yyyy
RegR:n:f	hodnota reálného registru se vypisuje dekadicky s f desetinnými místy, doplňuje se zleva na celkem n znaků mezerami

V jednom popisu **print** můžeme uvést více výše uvedených prvků, navzájem oddělených čárkou - viz řádka {4}.

Jestliže potřebujeme na jednom místě vypisovat proměnný text, závislý na hodnotě číselné proměnné či jinou obecnější proměnnou skupinu výpisů, použijeme k tomu popis **case**, uvedený v kap. 7.5.19

Příklad: DTEXT

7.5.6 Popis EDIT

Potřebujeme-li změnit hodnotu číselného nebo stringového registru, můžeme použít popis **EDIT**.

Tento popis má pro celočíselné registry tvar

```
EDIT reg:n, min, max;
```

například:

```
EDIT I100:5, 0, 1000;
```

Pro reálné registry má tvar

```
EDIT reg:n:f, min, max;
```

Pro registry typu string má tvar

```
EDIT reg : n;
```

Pro použití popisu **edit** platí jedno důležité pravidlo - smí být v popisu 1 obrazovky použit nanejvýš 1x, výjimkou je použití popisu **edit** uvnitř popisu **case**. V tomto případě může být popis **edit** použit v každé větvi popisu **case** viz kap. 7.5.19.

Tímto popisem uvádíme do činnosti editační režim, ve kterém se provádí editace hodnoty zadaného registru. Na obrazovku se vypíše (do místa, uvedeného v dříve uvedeném popisu **position** na celkem **n** posic, v případě reálného

registru s **f** desetinnými místy) stávající hodnota, kterou je možno u číselných registrů změnit v mezích **min** až **max** . Editace se končí klávesou Enter (potvrzení nové hodnoty) nebo klávesou ESC (opuštění editačního režimu beze změny hodnoty).

Po ukončení editace se provádí automaticky návrat na obrazovku, ze které byla obrazovka obsahující popis EDIT vyvolána.

Je-li v popisu obrazovky uveden popis **edit**, nemůžeme použít popis **onkey** a **wait**, stisknutí všech kláves se chápe jako editace hodnoty. Výjimku tvoří použití těchto popisů v jiných sekcích popisu **case**.

Příklad: DEDIT

7.5.7 Popis EDITP

Popis EDITP slouží pro zadávání hesla. Tvar příkazu je následující:

```
EDITP reg:n;
```

kde reg je registr typu string, který se edituje do délky n znaků. Při editaci se po stisku klávesy objeví pouze hvězdička, zadaný znak zůstává utajen. Ostatní vlastnosti příkazu jsou stejné jako u příkazu EDIT.

Příklad: DEDITP

7.5.8 Popis EDITENTER

Popis **EDITENTER** smí být v popisu obrazovky **TERMINAL** uveden jen ve spojení s příkazy **edit** a **editp** a to jen 1x. Příkaz má syntaxi

```
EDITENTER příkaz1;
```

Je-li tento popis uveden a skončí-li editace na základě popisu **EDIT**, resp. **EDITP**, stisknutím klávesy **Enter**, (potvrzení zeditované hodnoty), provede se příkaz **příkaz1** .

Na místě příkazu **příkaz1** smí být uveden libovolný příkaz vyjma příkazu skoku. Smí zde tedy být uveden i příkaz volání procedury, resp. složený příkaz, t.j. blok příkazů začínající klíčovým slovem **begin** a končící slovem **end**.

Tímto popisem můžeme nadefinovat činnosti, které je třeba naprogramovat jako přímou reakci na potvrzení změny hodnoty registru při jeho editaci.

Příklady: DEDIT, DEDITP

7.5.9 Popis RECT

Třetí vrstvu - vykreslení grafiky definujeme pomocí popisů grafických objektů.

Ve výše uvedeném příkladu je na řádce {5} uveden popis **rect**, kterým definujeme vykreslení obdélníku, zadaného horním levým a spodním pravým rohem.

Různé varianty popisu **RECT** jsou uvedeny v následující tabulce:

popis	význam
RECT x1, y1, x2, y2	vykreslení obdélníku, zadaného levým horním a pravým spodním rohem
RECT rel, x1, y1, w, h	vykreslení obdélníku, zadaného levým horním rohem, šířkou a výškou.

Příklad: DGRAF

7.5.10 Popis FILL

Popisem **FILL** definujeme vykreslení vyplněného obdélníku.

Různé varianty popisu **FILL** jsou uvedeny v následující tabulce:

popis	význam
FILL x1, y1, x2, y2	vykreslení vyplněného obdélníku, zadaného levým horním a pravým spodním rohem
FILL rel, x1, y1, w, h	vykreslení vyplněného obdélníku, zadaného levým horním rohem, šířkou a výškou.

Příklad: DGRAF

7.5.11 Popis LINE

Popisem **LINE** definujeme vykreslení úsečky.

Různé varianty popisu **LINE** jsou uvedeny v následující tabulce:

popis	význam
LINE x1, y1, x2, y2	vykreslení úsečky z bodu x1y1 do x2y2
LINE rel, x1, y1, dx, dy	vykreslení úsečky z bodu x1y1 do bodu, zadaného relativně vzhledem k x1y1

Příklad: DGRAF

7.5.12 Popis POINT

Popisem **`POINT`** definujeme vykreslení bodu:

popis	význam
POINT x1, y1	vykreslení bodu x1y1

Příklad: DGRAF

7.5.13 Popis CIRCLE

Popisem **`CIRCLE`** definujeme vykreslení kružnice:

Popis	význam
CIRCLE x1, y1, r	vykreslení kružnice, zadané středem a poloměrem

Příklad: DGRAF

7.5.14 Popis GRAPH

Popisem **`GRAPH`** definujeme vykreslení grafu, složeného z bodů, čar nebo sloupců, u kterého zadáváme pouze y-souřadnice jednotlivých bodů. V dále uvedené tabulce jsou použity tyto parametry:

N — počet bodů grafu. Na zadané grafické ploše bude 1. bod vykreslen na x-souřadnici xMin, N. tý bod na souřadnici xMax.

yData — registr s hodnotou pro první bod. Za tímto registrem musí následovat dalších N-1 registrů stejného typu, v kterých jsou umístěny hodnoty y pro další body. Hodnoty x pro všechny body jsou pevné a jsou rozděleny rovnoměrně přes celý x-rozměr grafické plochy.

yValMin — hodnota y-souřadnice bodu, příslušející spodní souřadnici y (yMax resp. yMin+yRel). Je-li hodnota bodu menší, nahrazuje se touto hodnotou a bod se vykresluje na spodní okraj grafické plochy.

yValMax — hodnota y-souřadnice bodu, příslušející horní souřadnici y (yMin). Je-li hodnota bodu větší, nahrazuje se touto hodnotou a bod se vykresluje na horní okraj grafické plochy.

xMin,yMin — definice horního levého rohu grafické plochy

xMax,yMax — definice spodního pravého rohu grafické plochy

xRel,yRel — definice velikosti grafické plochy

Na místě parametrů yValMin, yValMax, xMin, yMin, xMax, yMax, xRel, yRel můžeme uvést buď celočíselný konstantní výraz v rozsahu -1000 až 1000 (tj. jednoduchá konstanta, nebo výraz složený z více konstant, např. 100, K1, K1+100, K1+K2 atd.), nebo název registru (např. B0, I10, W100, L1000) nebo název registru s indexem. (např. B0[4], I10[B0], W100[W0], L1000[B0]). Nemůžeme zde však uvést obecný výraz (např. B0+1). V případě použití registru s indexem musí být index buď opět celočíselný konstantní výraz nebo název registru.

Chceme-li vyznačit na obrazovce okraje grafické plochy, musíme napsat navíc popis RECT se stejnými parametry xMin, yMin, xMax, yMax, resp. xMin, yMin, xRel, yRel.

Chceme-li na jednu plochu uvést přes sebe 2 grafy, stačí nadefinovat dva popisy **`GRAPH`** se stejnou velikostí a stejným umístěním grafické plochy. Jeden popis můžeme například zadat s parametrem **`line`**, druhý s parametrem **`point`**.

Různé varianty popisu **`GRAPH`** jsou uvedeny v následující tabulce:

Popis	význam
GRAPH point N, yData, yValMin, yValMax, xMin, yMin, xMax, yMax	vykreslení grafu, složeného z N bodů na zadané ploše. Plocha grafu je vymezena souřadnicemi levého horního xMin, yMin a pravého dolního rohu xMax, yMax.
GRAPH point rel N, yData, yValMin, yValMax, xMin, yMin, xRel, yRel	vykreslení grafu, složeného z N bodů na zadané ploše. Plocha grafu je vymezena souřadnicemi levého horního xMin, yMin a velikostí grafické plochy xRel a yRel.
GRAPH line N, yData, yValMin, yValMax, xMin, yMin, xMax, yMax	vykreslení grafu, složeného z N bodů, spojených úsečkami na zadané ploše. Plocha grafu je vymezena souřadnicemi levého horního xMin, yMin a pravého dolního rohu xMax, yMax.
GRAPH line rel N, yData, yValMin, yValMax, xMin, yMin, xRel, yRel	vykreslení grafu, složeného z N bodů spojených úsečkami na zadané ploše. Plocha grafu je vymezena souřadnicemi levého horního xMin, yMin a velikostí grafické plochy xRel a yRel.
GRAPH bar N, yData, yValMin, yValMax, xMin, yMin, xMax, yMax	vykreslení grafu, složeného z N svislých sloupců o výšce odpovídající hodnotě y-souřadnice bodu a šířce (xMax-xMin)/N na zadané ploše. Plocha grafu je vymezena souřadnicemi levého horního xMin, yMin a pravého dolního rohu xMax,

	yMax.
GRAPH bar rel N, yData, yValMin, yValMax, xMin, yMin, xRel, yRel	vykreslení grafu, složeného složeného z N svislých sloupců o výšce odpovídající hodnotě y-souřadnice bodu a šířce xRel/N na zadané ploše. Plocha grafu je vymezena souřadnicemi levého horního xMin, yMin a velikostí grafické plochy xRel a yRel.

bodový graf: *čárový graf:* *sloupcový graf:*

Příklad: DGRAPH

7.5.15 Popis GRAPHXY

Popisem **GRAPHXY** definujeme vykreslení grafu, složeného z bodů, čar nebo sloupců, u kterého zadáváme x- i y-souřadnice vynášených bodů. V dále uvedené tabulce jsou použity tyto parametry:

N počet bodů grafu. Na zadané grafické ploše bude 1. bod vykreslen na x-souřadnici xMin, N. tý bod na souřadnici xMax.

xData registr s hodnotou x-souřadnice pro první bod. Za tímto registrem musí následovat další N-1 registrů stejného typu, v kterých jsou umístěny hodnoty x-souřadnic pro další body.

yData registr s hodnotou y-souřadnice pro první bod. Za tímto registrem musí následovat další N-1 registrů stejného typu, v kterých jsou umístěny hodnoty y-souřadnic pro další body.

xValMin hodnota x-souřadnice bodu, příslušející levé souřadnici x (xMin) grafické plochy.

xValMax hodnota x-souřadnice bodu, příslušející pravé souřadnici x (xMax resp. xMin+xRel) grafické plochy.

yValMin hodnota y-souřadnice bodu, příslušející spodní souřadnici y (yMax resp. yMin+yRel) grafické plochy.

yValMax hodnota y-souřadnice bodu, příslušející horní souřadnici y (yMin) grafické plochy.

xMin,yMin definice horního levého rohu grafické plochy

xMax,yMax definice spodního pravého rohu grafické plochy

xRel,yRel definice velikosti grafické plochy

Na místě parametrů xValMin, xValMax, yValMin, yValMax, xMin, yMin, xMax, yMax, xRel, yRel můžeme uvést buď celočíselný konstantní výraz v rozsahu -1000 až 1000 (tj. jednoduchá konstanta, nebo výraz složený z více konstant, např. 100, K1, K1+100, K1+K2 atd.), nebo název registru (např. B0, I10, W100, L1000) nebo název registru s indexem. (např. B0[4], I10[B0], W100[W0], L1000[B0]). Nemůžeme zde však uvést obecný výraz (např. B0+1). V případě použití registru s indexem musí být index buď opět celočíselný konstantní výraz nebo název registru.

Chceme-li vyznačit na obrazovce okraje grafické plochy, musíme napsat navíc popis **RECT** se stejnými parametry xMin, yMin, xMax, yMax, resp. xMin, yMin, xRel, yRel.

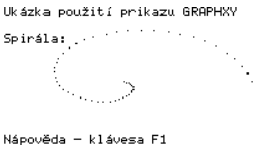
Chceme-li na jednu plochu uvést přes sebe 2 grafy, stačí nadefinovat dva popisy **GRAPHXY** se stejnou velikostí a umístěním grafické plochy. Jeden popis můžeme například zadat s parametrem **line**, druhý s parametrem **point**.

Různé varianty popisu **GRAPHXY** jsou uvedeny v následující tabulce:

popis	Význam
GRAPHXY point N, xData, yData xValMin,xValMax, yValMin,yValMax, xMin, yMin, xMax, yMax	vykreslení grafu, složeného z N bodů na zadané ploše. Plocha grafu je vymezena souřadnicemi levého horního xMin, yMin a pravého dolního rohu xMax, yMax.
GRAPHXY point rel N, xData, yData xValMin, xValMax,yValMin, yValMax, xMin, yMin, xRel, yRel	vykreslení grafu, složeného z N bodů na zadané ploše. Plocha grafu je vymezena souřadnicemi levého horního xMin, yMin a velikostí grafické plochy xRel a yRel.
GRAPHXY line N, xData, yData, xValMin,xValMax, yValMin,yValMax, xMin, yMin, xMax, yMax	vykreslení grafu, složeného z N bodů, spojených úsečkami na zadané ploše. Plocha grafu je vymezena souřadnicemi levého horního xMin, yMin a pravého dolního rohu xMax, yMax.
GRAPHXY line rel N, xData, yData, xValMin, xValMax,yValMin, yValMax,yMin, xRel, yRel	vykreslení grafu, složeného z N bodů spojených úsečkami na zadané ploše. Plocha grafu je vymezena souřadnicemi levého horního xMin, yMin a velikostí grafické plochy xRel a yRel.
GRAPHXY bar N, xData, yData, xValMin,xValMax, yValMin,yValMax, xMin, yMin, xMax, yMax	vykreslení grafu, složeného z N obdélníků, zadaných protilehlými rohy vždy dle dvou po sobě jdoucích bodů na zadané ploše. Plocha grafu je vymezena souřadnicemi levého horního xMin, yMin a pravého dolního rohu xMax, yMax.

GRAPHXY bar rel N, xData, yData, xValMin, xValMax,yValMin, yValMax, xMin, yMin, xRel, yRel	vykreslení grafu, složeného z N obdélníků, zadaných protilehlými rohy vždy dle dvou po sobě jdoucích bodů na zadané ploše. Plocha grafu je vymezena souřadnicemi levého horního xMin, yMin a velikostí grafické plochy xRel a yRel.

bodový graf:



čárový graf:

sloupcový graf:

Příklad: DGRAPHXY

7.5.16 Popis BAR

Popisem **BAR** definujeme vykreslení bar-grafu, tj. částečně vyplněné grafické oblasti na základě skutečné hodnoty parametrů Val, ValMin a ValMax.

V dále uvedené tabulce jsou použity tyto parametry:

Val	registr se zobrazovanou hodnotou.
ValMin	hodnota, příslušející počáteční hraně bargrafu. Je-li hodnota registru menší, je bargraf zcela prázdný.
ValMax	hodnota, příslušející koncové hraně bargrafu. Je-li hodnota registru větší, je bargraf zcela zaplněn.
xMin,yMin	definice horního levého rohu plochy bargrafu
xMax,yMax	definice spodního pravého rohu plochy bargrafu
xRel,yRel	definice velikosti plochy bargrafu

Na místě parametrů Val, ValMin, ValMax, xMin, yMin, xMax, yMax, xRel, yRel můžeme uvést buď celočíselný konstantní výraz v rozsahu -1000 až 1000 (tj. jednoduchá konstanta, nebo výraz složený z více konstant, např. 100, K1, K1+100, K1+K2 atd.), nebo název registru (např. B0, I10, W100, L1000) nebo název registru s indexem. (např. B0[4], I10[B0], W100[W0], L1000[B0]). Nemůžeme zde však uvést obecný výraz (např. B0+1), v případě použití registru s indexem musí být index buď opět celočíselný konstantní výraz nebo název registru.

Chceme-li vyznačit na obrazovce okraje bargrafu, musíme napsat navíc popis RECT se stejnými parametry xMin, yMin, xMax, yMax resp. xMin, yMin, xRel, yRel.

Různé varianty popisu **BAR** jsou uvedeny v následující tabulce:

Popis	význam
BAR bottom Val, ValMin, ValMax, xMin, yMin, xMax, yMax	vykreslení bargrafu na zadané ploše. ValMin odpovídá spodní hraně, ValMax horní hraně bargrafu. Plocha bargrafu je vymezena souřadnicemi levého horního xMin, yMin a pravého dolního rohu xMax, yMax.
BAR top Val, ValMin, ValMax, xMin, yMin, xMax, yMax	vykreslení bargrafu na zadané ploše. ValMin odpovídá horní hraně, ValMax spodní hraně bargrafu. Plocha bargrafu je vymezena souřadnicemi levého horního xMin, yMin a pravého dolního rohu xMax, yMax.
BAR left Val, ValMin, ValMax, xMin, yMin, xMax, yMax	vykreslení bargrafu na zadané ploše. ValMin odpovídá levé hraně, ValMax pravé hraně bargrafu. Plocha bargrafu je vymezena souřadnicemi levého horního xMin, yMin a pravého dolního rohu xMax, yMax.
BAR right Val, ValMin, ValMax, xMin, yMin, xMax, yMax	vykreslení bargrafu na zadané ploše. ValMin odpovídá pravé hraně, ValMax levé hraně bargrafu. Plocha bargrafu je vymezena souřadnicemi levého horního xMin, yMin a pravého dolního rohu xMax, yMax.
BAR bottom rel Val, ValMin, ValMax, xMin, yMin, xRel, yRel	vykreslení bargrafu na zadané ploše. ValMin odpovídá spodní hraně, ValMax horní hraně bargrafu. Plocha bargrafu je vymezena souřadnicemi levého horního xMin, yMin a velikostí xRel a yRel.
BAR top rel Val, ValMin, ValMax, xMin, yMin, xRel, yRel	vykreslení bargrafu na zadané ploše. ValMin odpovídá horní hraně, ValMax spodní hraně bargrafu. Plocha bargrafu je vymezena souřadnicemi levého horního xMin, yMin a velikostí xRel a yRel.
BAR left rel Val, ValMin, ValMax, xMin, yMin, xRel, yRel	vykreslení bargrafu na zadané ploše. ValMin odpovídá levé hraně, ValMax pravé hraně bargrafu. Plocha bargrafu je vymezena souřadnicemi levého horního xMin, yMin a velikostí xRel a yRel.
BAR right rel Val, ValMin, ValMax, xMin, yMin, xRel, yRel	vykreslení bargrafu na zadané ploše. ValMin odpovídá pravé hraně, ValMax levé hraně bargrafu. Plocha bargrafu je vymezena souřadnicemi levého horního xMin, yMin a velikostí xRel a yRel.

Příklad: DBAR

7.5.17 Popis ONKEY (reakce na stisk kláves)

K popisu reakce na stisk kláves (nejedná-li se o obrazovku, která obsahuje popis **edit**), slouží příkazy **onkey** a **wait**.

Příkaz **onkey** se zpravidla uvádí v deklaraci TERMINAL jen 1x. Příkaz má syntaxi

```
ONKEY
 keyvalue1 : příkaz1;
 keyvalue2 : příkaz2;
 ...
 keyvalueN : příkazN;
END;
```

Je-li stisknuta jedna z kláves **keyvaluex**, provede se jeden z příslušných příkazů **příkaz1** až **příkazN**, v případě stisku klávesy, která zde není uvedena, se neprovede žádná činnost.

Na místě **keyvaluex** můžeme zapsat buď jednu konstantu ve tvaru

```
keyvalue
```

nebo více konstant oddělených čárkou ve tvaru

```
keyvalue1,keyvalue2,..,keyvaluek
```

Chceme-li popsat reakci na stisk speciálních kláves, použijeme pro označení těchto kláves následující identifikátory:

identifikátor	Specifikaceklávesy
ZCR	ENTER
ZBS	BACK SPACE
zESC	ESC
zSPACE	SPACE (mezerník)
zDEL	DEL
ZUP	↑
ZDn	↓
ZLe	←
ZRi	→
zF1 až zF10	F1 až F10
zINS	INS
zTAB	TAB
zPgUp	SHIFT + ↑
zPgDn	SHIFT + ↓
zHome	SHIFT + ←
zEnd	SHIFT + →

Ostatní klávesy se popisují jednoduše jejich ASCII hodnotou v apostrofech. Např. : 'R', '1', '$'

Pozor: klávesy F1 a ESC mají speciální význam, na stisk klávesy F1 dojde k vyvolání obrazovky HELP viz. 7.5.20, na stisk klávesy ESC se vrátíme do předchozí obrazovky. Po stisku klávesy ESC se nejdříve nastaví předchozí číslo obrazovky a pak se provede uživatelem nadefinovaná obsluha.

Hodnota **keyvaluex** se v jednotlivých sekcích příkazu **onkey** nesmí opakovat, vyjímkou je opakování v paraleně zařazených větvích popisu **CASE**.

Na místě příkazu **příkazx** smí být uveden libovolný příkaz vyjma příkazu skoku. Smí zde tedy být uveden i příkaz volání procedury, resp. složený příkaz, t.j. blok příkazů začínající klíčovým slovem **begin** a končící slovem **end**.

Příklad: DONKEY

7.5.18 Popis WAIT

Popis **wait** smí být v deklaraci TERMINAL uveden jen 1x. Příkaz má syntaxi

```
WAIT timeout, nové číslo obrazovky;
```

Pokud tento příkaz použijeme, znamená to, že pokud po dobu **timeout** nedojde ke stisku žádné klávesy, přejdeme automaticky na obrazovku „**nové číslo obrazovky**". Tento příkaz nám pomáhá k tomu, abychom na dané obrazovce nesetrvali příliš dlouho. Popis **WAIT** můžeme kombinovat s popisem **ONKEY**, takže pokud chceme na stisk nějaké klávesy vyskočit dříve, je třeba použít popisu **ONKEY**.

Příklad: DWAIT

7.5.19 Popis CASE

Někdy potřebujeme provést výpis na obrazovku variantně na základě hodnoty nějakého registru. K tomu může posloužit popis **CASE**. Ten má definici:

```
CASE Reg OF
```

```
 hodnota1 : popis1;
 hodnota2 : popis2;
 ...
 hodnotaN : popisN;
 ELSE popisE;
END;
```

Hodnota registru Reg (který smí být typu byte nebo bit) je testována na rovnost s konstantními hodnotami **hodnota1** až **hodnotaN.** V případě shody se provede výpis na obrazovku dle příslušné větve popisu case, tj. jednoho z příslušných popisů **popis1** až **popisN**. V případě neúspěchu u všech testovaných hodnot se na obrazovku provede výpis na základě **popisE** za klíčovým slovem **ELSE**. Není-li část s klíčovým slovem **ELSE** zapsána, neprovede se v tomto případě žádný výpis.

Na místě **hodnotax** můžeme zapsat buď jednu konstantu ve tvaru

const

nebo více konstant oddělených čárkou ve tvaru

const1,const2,...,constk

nebo interval konstant ve tvaru

const1..const2

nebo několik intervalů konstant ve tvaru

const1..const2,const3..const4

Výše uvedené konstanty musí být typu byte, resp. bit, ve shodě s typem registru Reg

Na místě **popisx** můžeme uvést libovolný popis z první, druhé nebo třetí vrstvy vyjma popisu **CASE** (popis CASE nelze vnořovat do sebe) nebo složený popis, který má tvar:

begin popis1; popis2; …popisn **end**.

7.5.20 Popis obrazovky HELP

Ke každé uživatelské obrazovce lze jednoduše napsat párovou HELP-obrazovku, kterou vyvoláme stisknutím klávesy F1.

Popis toho, co má být uvedeno na HELP obrazovce zapíšeme do deklarace TERMINAL za klíčové slovo **help** - viz příklad z kap. 7.5, řádka {10} a {11}.

HELP-obrazovku opustíme stisknutím klávesy ESC resp. ENTER.

Pro popis HELP obrazovky můžeme použít veškeré popisy vyjma popisů **edit**, **editenter**, **editp**, **onkey** a **wait**.

Obsah HELP-obrazovky se generuje staticky při prvním ošetření dané obrazovky, tzn. veškeré proměnné prvky (hodnoty registrů, proměnné souřadnice atd.) jsou v HELP-obrazovce neměnné.

Příklad: DTEXT

7.6 Použití prvků pro jednotlivé typy terminálů

Jednotlivé popisy smíme u jednotlivých typů terminálů použít v tomto rozsahu:

	TERM10	TERM01	GENSTR
Bitmap	*		
Position	*	*	
Font	*		
Print	*	*	*
Point	*		
Line	*		
Rect	*		
Fill	*		
Circle	*		
Bar	*		
Graph	*		
GraphXY	*		

Case	*	*	*
OnKey	*	*	
Wait	*	*	
Edit	*	*	
Editp	*	*	
EditEnter	*	*	
Help	*	*	

8. Vstupy a výstupy

V této kapitole jsou popsány HW-objekty obsluhující jednotlivé desky vstupů a výstupů, které mohou být v systému obsaženy.

8.1 Úvod

Samotná procesorová deska ve většině aplikací nenachází uplatnění, desku je třeba rozšířit o vstupy a výstupy. V prostředí KIT-BUILDER je ke každému typu V/V-desky (typu V/V portu) přidělen tzv. HW objekt umožňující nastavit jednotlivé parametry daného vstupu/výstupu.

V úvodu jsou popsány základní prvky používané pro všechny V/V, v dalších kapitolách je konkrétní popis pro daný typ V/V desky.

Parametr	Význam parametru
HWOBJ	Následuje označení typu desky
NAME	Definuje prefix symbolického jména, pod kterým je možno k jednotlivým portům přistupovat. Pokud není uvedeno, bere se označení typu desky. Pokud jsou použity dvě desky stejného typu, je třeba parametr NAME u druhé a další desky použít.
ADR	HW adresa připojené desky, její hodnota je určena nakonfigurováním desky.
VAR	Adresa uživatelských registrů, do kterých jsou mapovány jednotlivé vstupy a výstupy. Adresa musí být správného typu (Byte u binárních, Integer u analogových vstupů a výstupů). Pokud se tento parametr neuvede, jsou potřebné registry automaticky přiděleny jako u symbolické definice proměnných.
MAIN	Určuje, které vstupy a výstupy budou obsluhovány na začátku a konci cyklu MAIN.
FAST	Určuje, které vstupy a výstupy budou obsluhovány na začátku a konci cyklu FAST.
CASE	Určuje skupinu vstupů a výstupů, z kterých je vždy jeden cyklicky obsloužen na konci cyklu MAIN.
NOT	Uvedením parametru NOT před jménem používaného binárního portu můžeme přepnout obsluhu portu z pozitivní do negativní logiky.

Jako příklad použití jednotlivých parametrů si prostudujte kapitolu 8.2 Typ IOPBUS, u ostatních objektů jsou již uváděny pouze symbolické názvy jednotlivých vstupních a výstupních portů, syntaxe je obdobná.

8.2 Typ IOPBUS

```
CONFIGURATION
 HWOBJ=IOPBUS,NAME=KITPBUS,ADR=$d220,VAR=B500:3,
       MAIN=[INA,OUTC],C=$AA;
```

Objekt popisuje trojici binárních vstupů/výstupů vyskytujících se např. na základní procesorové desce KITV40.

Na uvedeném příkladu je popsán IOPBus na procesorové desce KITV40, který má standardní HW adresu $D220, (parametr **ADR=$d220**), z této sběrnice budeme používat port A jako vstupní, port B nebudeme používat vůbec a port C budeme používat jako výstupní. Oba porty A i C budou čteny, resp. zapisovány, v režimu MAIN, (parametr **MAIN=[INA,OUTC]**). Dále předepisujeme, že po resetu má mít port-C inicializační hodnotu $AA (parametr **C=$aa**).

V tomto případě není využito automatické přidělení uřivatelských registrů pro přístup k objektu a využívá se jejich ručního definování. Parametr **VAR=B500:3** pak říká, že porty A, B a C jsou mapovány do 3 uživatelských registrů počínaje registrem B500, tj. registrů B500 až B502. Z registru B500 můžeme číst vstup z portu A, do registru B502 ukládáme hodnotu, která se má zapsat na výstupní port C. Registr B501 je v tomto případě nevyužit, protože port B není aktivován ani jako vstup ani jako výstup. Údaj **:3** je při definici nepovinný, doporučuje se však jej uvádět, aby měl uživatel přehled o počtu bytů, které tento HW objekt mapuje.

Parametr **NAME=KITPBUS** specifikuje název tohoto HW-objektu. Tento symbolický název objektu můžeme použít ve spojení se symbolickým názvem jednotlivých položek (zde A, B resp. C) k symbolickému označení registrů B500 až B502 takto:

KITPBUS_A odpovídá B500,

KITPBUS_B odpovídá B501,

KITPBUS_C odpovídá B502.

Procedura MAIN, která pouze přepisuje negovaný port A na port C pak může mít tvar:

```
procedure MAIN;
begin
 KITPBUS_C:=not KITPBUS_A;
end;
```

Pokud při deklaraci objektu nepoužijeme parametr NAME, je takový objekt pojmenován stejně jako jeho typ, tj. v tomto případě IOPBUS. Vzhledem k tomu, že žádný identifikátor nesmí být v programu definován 2x, by se však hlásila chyba, kdybychom chtěli nadefinovat objekt IOPBUS vícekrát a nepoužili (vyjma jedné definice) parametr NAME.

Pokud bychom potřebovali číst, resp. zapisovat, port synchronně každých 10ms, můžeme místo parametru MAIN=[..] použít parametr FAST=[..]. V takovém případě se příslušný vstupní port čte vždy před vyvoláním uživatelské procedury FAST, příslušný výstupní port zapisuje po vyvolání této procedury. Vždy mějme na paměti, že proces FAST by měl být co nejkratší, tj. této možnosti bychom měli využít jen v případě nutnosti.

Stručně můžeme objekt IOPBUS popsat následující tabulkou:

Offset uživatelského registru vzhledem k adrese VAR	*Definice portu v parametru MAIN/FAST*	*Označení portu při symbolickém adresování*	*Význam*
0	INA/OUTA	A	Port A
1	INB/OUTB	B	Port B
2	INC/OUTC	C	Port C

Takovouto tabulkou popíšeme v následujicích kapitolách i další desky vstupů a výstupů.

Příklad: DIOPBUS

8.3 Typ IODIO01

```
CONFIGURATION
 HWOBJ=IODIO01,ADR=$2310,FAST=[INA,INB,INC,IND];
```

Objekt popisuje čtveřici binárních vstupů, vyskytujících se např. na desce IODIO01.

Offset uživatelského registru vzhledem k adrese VAR	*Definice portu v parametru MAIN/FAST*	*Označení portu při symbolickém adresování*	*Význam*
0	INA	A	Port A
1	INB	B	Port B
2	INC	C	Port C
3	IND	D	Port D

Příklad: DIODIO01

8.4 Typ IODOO01

```
CONFIGURATION
 HWOBJ=IODOO01,ADR=$2320,MAIN=[OUTA,OUTB];
```

Objekt popisuje čtveřici binárních výstupů, vyskytujících se např. na desce IODOO01.

Offset uživatelského registru vzhledem k adrese VAR	*Definice portu v parametru MAIN/FAST*	*Označení portu při symbolickém adresování*	*Význam*
0	OUTA	A	Port A
1	OUTB	B	Port B
2	OUTC	C	Port C
3	OUTD	D	Port D

Příklad: DIODOO01

8.5 Typ IODXO01

```
CONFIGURATION
 HWOBJ=IODXO01,ADR=$2330,FAST=[INA,OUTC],
       MAIN=[INB,OUTD];
```

Objekt popisuje čtveřici binárních vstupů resp. výstupů, vyskytujících se např. na desce IODXO01 (A, B povinně vstup, C, D povinně výstup).

Offset uživatelského registru vzhledem k adrese VAR	*Definice portu v parametru MAIN/FAST*	*Označení portu při symbolickém adresování*	*Význam*
0	INA	A	Port A
1	INB	B	Port B
2	OUTC	C	Port C
3	OUTD	D	Port D

Příklad: DIODXO01

8.6 Typ IOTERM10

```
CONFIGURATION
 HWOBJ=IOTERM10,ADR=$2300,MAIN=[IN,OUT];
```

Objekt popisuje port binárních vstupů a port binárních výstupů umístěných např. na desce TERM10.

Offset uživatelského registru vzhledem k adrese VAR	*Definice portu v parametru MAIN/FAST*	*Označení portu při symbolickém adresování*	*Význam*
0	IN	IN	Port IN
1	OUT	OUT	Port OUT

Příklad: DIOT10

8.7 Typ IODTERM10

```
CONFIGURATION
 HWOBJ=IODTERM10,ADR=$2310,FAST=[INA,OUTC],
       MAIN=[INB,OUTD];
```

Objekt popisuje čtveřici binárních vstupů, resp. výstupů, vyskytujících se na desce IOTERM10 (A, B povinně vstup, C, D povinně výstup).

Offset uživatelského registru vzhledem k adrese VAR	*Definice portu v parametru MAIN/FAST*	*Označení portu při symbolickém adresování*	*Význam*
0	INA	A	Port A
1	INB	B	Port B
2	OUTC	C	Port C
3	OUTD	D	Port D

8.8 Typ IOATERM10

```
CONFIGURATION
 HWOBJ=IOATERM10,NAME=ADT,ADR=$2310,
       VAR=I400:46,CASE=[IN0=BIT10,IN1=BIT14,
       IN2=PT100:-100:1000,OUT0,TEMP=5],
       OUT0=1000;
```

Objekt popisuje analogovou část desky IOTERM10. Tato deska má celkem max. 16 analogových vstupů (IN0 až IN15), 6 analogových výstupů (OUT0 až OUT5) a vstup TEMP pro čtení teploty připojovacích konektorů.

Rozsahy vstupních, resp. výstupních signálů jednotlivých vstupů, resp. výstupů jsou dány HW zapojením na desce. Přesné hodnoty rozsahů, případně popis jejich nastavení se dočtete přímo v manuálu k desce IOTERM10.

Pro čtení **analogových vstupů** můžeme v závislosti na konkrétním vstupu zvolit některý z následujících režimů

BIT8, BIT10, BIT12, BIT14 BIT16 - signál připojený na svorky převodníku je měřen s danou přesností (pozn. čím větší přesnost, tím delší doba měření).

PT100 – lze použít pokud analogový vstup umožňuje přímo připojit odporové čidlo teploty PT100. Následující parametry -100 a 1000 udávají pracovní rozsah čidla (implicitně -50°C až 200°C). Výstupem převodníku je hodnota v rozsahu -999.9°C až 999.9°C s přesností 0.1°C.

Navolení režimu pro vstupní A/D kanál zapisujeme v podobě uvedené na příkladu.

Analogové výstupy jsou 12 bitové tj. v rozsahu 0-4095, čemuž odpovídá výstupní hodnota podle hardwarové konfigurace propojek umístěných přímo na desce.

Dále je na desce připojen speciální vstup TEMP, umožňující měřit teplotu připojovacích konektorů, parametr, kterým se doplňuje říká, jak často se má číst tato teplota v sekci CASE.

Sekce CASE je sekce určená pro obsluhu pomalých vstupů a výstupů. Všechny vstupy a výstupy desky IOTERM10 mohou být zařazeny pouze do této sekce. Zpracování probíhá tak, že při každém systémovém volání se obslouží pouze jedna z položek v sekci CASE, položka TEMP se obsluhuje až při jejím N-tém volání, kde parametr N zadáváme při konfiguraci vstupu TEMP.

Na uvedeném příkladu je popsáno připojení IOATERM10 desky, která má standardní HW adresu $2310, (parametr **ADR=$2310**), z této desky budeme používat vstup IN0 a IN1 s přesností 10 a 14 bitů a vstup IN2, na kterém je připojeno čidlo teploty PT100. Ostatní vstupy nebudeme používat vůbec. Výstupní kanál používáme pouze OUT0. Dále je definován vstup TEMP. Všechny kanály budou čteny, resp. zapisovány, v režimu CASE (parametr **CASE=[...]**) tj. čtení vstupních kanálů i zápis na výstupní kanál OUT0 se bude provádět cyklicky vždy při vykonávání systémových služeb, parametr 5 u vstupu TEMP říká, že vstup TEMP bude obsluhován 5x pomaleji než ostatní položky v sekci CASE.

Dále předepisujeme, že po resetu má mít výstupní kanál inicializační hodnotu 1000 (parametr **OUT0=1000**).

Nepovinný (pokud není uveden, registry jsou přiděleny automaticky) parametr **VAR=I400:46** říká, že IN a OUT kanály jsou mapovány do integer registrů počínaje registrem I400, celková délka vyhrazeného prostoru činí 46 bytů, tj. 23 integer registrů. Údaj **:46** je při definici nepovinný, doporučuje se však jej uvádět, aby měl uživatel přehled o počtu bytů, které tento HW objekt mapuje.

Přiřazení jednotlivých registrů jednotlivým kanálům je uvedeno v tabulce:

Absolutní adresa registru	kanál	povolený režim pro stand. HW	symbolický název registru v příkladu
I400	IN0	BIT8-16	ADT_IN0
I402	IN1	BIT8-16	ADT_IN1
I404	IN2	BIT8-16	ADT_IN2
I406	IN3	BIT8-16	ADT_IN3
I408	IN4	BIT8-16	ADT_IN4
I410	IN5	BIT8-16	ADT_IN5
I412	IN6	BIT8-16	ADT_IN6
I414	IN7	BIT8-16	ADT_IN7
I416	IN8	BIT8-16, PT100	ADT_IN8
I418	IN9	BIT8-16, PT100	ADT_IN9
I420	IN10	BIT8-16, PT100	ADT_IN10
I422	IN11	BIT8-16, PT100	ADT_IN11
I424	IN12	BIT8-16, PT100	ADT_IN12
I426	IN13	BIT8-16, PT100	ADT_IN13
I428	IN14	BIT8-16, PT100	ADT_IN14
I430	IN15	BIT8-16, PT100	ADT_IN15
I432	OUT0		ADT_OUT0
I434	OUT1		ADT_OUT1
I436	OUT2		ADT_OUT2
I438	OUT3		ADT_OUT3
I440	OUT4		ADT_OUT4
I442	OUT5		ADT_OUT5
I444	TEMP	-	ADT_TEMP

Parametr **NAME=ADT** specifikuje název tohoto HW-objektu. Tento symbolický název objektu můžeme použít ve spojení se symbolickým názvem jednotlivých položek k symbolickému označení registrů I400 až I444 tak, jak je uvedeno v posledním sloupci předchozí tabulky.

Pokud při deklaraci objektu nepoužijeme parametr NAME, je takový objekt pojmenován stejně jako jeho typ, tj. v tomto případě IOATERM10.

8.9 Typ IOADDA01

```
CONFIGURATION
 HWOBJ=IOADDA01, NAME=AD, ADR=$2300,
       VAR=I400:20,MAIN=[IN0=SU,IN1=SB,
       IN2=DU,IN4=DB,OUT0], OUT0=1000;
```

Objekt popisuje desku IOADDA01. Tato deska má celkem 8 analogových vstupů (IN0 až IN7) a 2 analogové výstupy (OUT0,OUT1).

Analogové vstupy mohou pracovat ve čtyřech režimech:

označení	význam
SU	singulární analogový vstup, digitální výstup v rozsahu 0 … 4095
SB	singulární analogový vstup, digitální výstup v rozsahu -4095 ... 4095
DU	diferenciální analogový vstup (rozdíl kanálů x a x+1 (x=0,2,4,6), kanál x+1 již nesmíme samostatně definovat), digitální výstup v rozsahu 0 … 4095
DB	diferenciální analogový vstup (rozdíl kanálů x a x+1 (x=0,2,4,6), kanál x+1 již nesmíme samostatně definovat), digitální výstup v rozsahu –4095 ... 4095

Navolení režimu pro vstupní A/D kanál zapisujeme v podobě uvedené na příkladu.

Analogové výstupy jsou 12ti bitové, vždy v digitálním rozsahu 0…4095. Vstupní digitální hodnoťě pak odpovídá výstupní analogová hodnota daná typem osazeného analogového výstupu (viz. katalogový list desky).

Na uvedeném příkladu je popsáno připojení IOADDA01 desky, která má standardní HW adresu $2300, (parametr **ADR=$2300**), z této desky budeme používat vstup IN0 a IN1 jako singulární, vstupy IN2 (spolu s IN3) a IN4 (spolu s IN5) jako diferenciální. Vstupní kanály IN6 a IN7 nebudeme používat vůbec. Z dvou možných výstupních kanálů OUT0 a OUT1 budeme používat kanál OUT0. Všechny kanály budou čteny, resp. zapisovány, v režimu MAIN, (parametr **MAIN=[...]**), tj. čtení ze vstupních kanálů se bude provádět cyklicky vždy před vstupem do uživatelské procedury

MAIN, zápis na výstupní kanál OUT0 se bude provádět cyklicky vždy po provedení uživatelské procedury MAIN. Dále předepisujeme, že po resetu má mít výstupní kanál inicializační hodnotu 1000 (parametr **OUT0=1000**).

Nepovinný (pokud není uveden, registry se přidělí automaticky) parametr **VAR=I400:20** říká, že IN a OUT kanály jsou mapovány do integer registrů počínaje registrem I400, celková délka vyhrazeného prostoru činí 20 bytů, tj. 10 integer registrů. Údaj **:20** je při definici nepovinný, doporučuje se však jej uvádět, aby měl uživatel přehled o počtu bytů, které tento HW objekt mapuje.

Přiřazení jednotlivých registrů jednotlivým kanálům je uvedeno v tabulce:

absolutní adresa registru	kanál	Povolený režim	symbolický název registru v příkladu
I400	IN0	S,D	AD_IN0
I402	IN1	(S)	AD_IN1
I404	IN2	S,D	AD_IN2
I406	IN3	(S)	AD_IN3
I408	IN4	S,D	AD_IN4
I410	IN5	(S)	AD_IN5
I412	IN6	S,D	AD_IN6
I414	IN7	(S)	AD_IN7
I416	OUT0		AD_OUT0
I418	OUT1		AD_OUT1

Parametr **NAME=AD** specifikuje název tohoto HW-objektu. Tento symbolický název objektu můžeme použít ve spojení se symbolickým názvem jednotlivých položek k symbolickému označení registrů I400 až I418 tak, jak je uvedeno v posledním sloupci předchozí tabulky.

Pokud při deklaraci objektu nepoužijeme parametr NAME, je takový objekt pojmenován stejně jako jeho typ, tj. v tomto případě IOADDA01. Vzhledem k tomu, že žádný identifikátor nesmí být v programu definován 2x, došlo by při druhém a dalším definování objektu IOADDA01 bez uvedeni odlišného parametru NAME automaticky k nahlášení chyby.

Příklad: DIOADDA

8.10 Typ IOFLEXPOS

```
CONFIGURATION
 HWOBJ=IOFLEXPOS, ADR=$2310,
 HWOBJ=IOFLEXPOS,NAME=FLEX,VAR=L100:29                    ,ADR=$2310, FAST;
```

Objekt slouží ve spojení s deskou IOFLEX01 k odměřování polohy. Po startu aplikace se do této desky nahraje firmware umožňující odměřování až pomocí čtyř snímačů.

Objekt může pracovat ve dvou režimech: Pokud je v definicici objektu uvedeno klíčové slovo FAST, pak se poloha čte každých 10, 20 nebo 50 ms před procedurou FAST. Jinak se obsluha desky IOFLEX01 provádí před provedením procedury MAIN.

Následující tabulka popisuje rozmístění uživatelských registrů tohoto objektu:

Offset uživatelského registru vzhledem k adrese VAR	*typ registru*	*Označení portu při symbolickém adresování*	*Význam*
Base + 0	Longint	TIME	čas v μs od posledního měření
Base + 4	Longint	POS0	hodnota snímače č.0
Base + 8	Longint	POS1	hodnota snímače č.1
Base + 12	Longint	POS2	hodnota snímače č.2
Base + 16	Longint	POS3	hodnota snímače č.3
Base + 20	Byte	STATUS0	stav snímače č.0
Base + 21	Byte	STATUS1	stav snímače č.1
Base + 22	Byte	STATUS2	stav snímače č.2
Base + 23	Byte	STATUS3	stav snímače č.3
Base + 24	Byte	CMD0	přikaz snímače č.0
Base + 25	Byte	CMD1	přikaz snímače č.1
Base + 26	Byte	CMD2	přikaz snímače č.2
Base + 27	Byte	CMD3	přikaz snímače č.3
Base + 28	Byte	APPLYCMD	potvrení příkazu

Registrům příkazů lze přiřadit následující hodnoty:

0 – běžný stav (= prázdná operace, pokud deska snímače nejsou ve stavu čekání na vynulování.)

1 – okamžité nulování (kalibrování čidel)

2 – nulování podmíněné průchodem nulou

3 – nulování podmíněné průchodem nulou a externím s signálem (viz. manuál k desce IOFLEX01)

Příkazy jsou prováděny pro všechny snímače současně. Jsou aktivovány zápisem hodnoty 1 do registry ApplyCmd. Po provedení přikazu je tento registr automaticky vynulován.

Stavové registry obsahuji trvale hodnotu 1. Pouze v případě, že byl na konkrétní snímač vyslán příkaz podmíněného nulování, je příslušný registr vynulován. V okamžiku splnění podmínky nulování je tento registr nastaven zpět na hodnotu 1.

Příklad: DIOFLEXP

8.11 Seznam HW desek a jejich ovladačů

V následující tabulce je přehled jednotlivých HW desek, resp HW sestav doplňených o jména typů ovladačů umožňujících jejich obsluhu.

Název desky	*Název ovladače*
KITV40	IOPBUS, Port C pouze OUT
KIT386EXR	IOPBUS
IOP	IOPBUS
IOPCOM	IOPBUS
IOP485I	IOPBUS
IODIO01	IODIO01
IODOO01,2,3	IODOO01
IODXO01,2	IODXO01
IOTERM10	IODTERM10 {digitální V/V} IOATERM10 {analogové V/V}
IOADDA01	IOADDA01
IOFLEX01	POSMEAS
IOTERM10	IOATERM10 IODTERM10
PDI040	IOPBUS, Port A a B IN, Port C Out IOTERM10
TERM10	IOTERM10 {V/V na TERM10}
KOMPAKT1	IOPBUS, Port A a B IN, Port C Out IOTERM10
KOMPAKT2	IOATERM10 IODTERM10
KOMPAKT3	IOATERM10 IODTERM10 IOPBUS, Port A a B IN, Port C Out IOTERM10

9. Komunikační linky

9.1 Úvod

V této kapitole je popsána kompletní obsluha komunikačních linek v prostředí KIT-BUILDER.

Pro obsluhu komunikační linky slouží tzv. HWOBJ s názvem COM. Jeho uvedením v konfigurační sekci CONFIGURATION programu provedeme jeho zařazení do obsluhovaných zařízení což nám dále v programu dovolí volat jednotlivé funkce a procedury pracující nad tímto zařízením.

Stavebnice KIT umožňuje prostřednictvím následujících ovladačů obsluhovat komunikační linky umístěné na následujících komponentech: IOPCOM, IOCOM,IO485I a IOTERM. Komunikační linka umístěná na desce KITV40 je určena pouze pro systémové funkce (program KBDCON), při kterých se využívá otevřeného protokolu firmy SofCon.

9.2 Konfigurace

```
CONFIGURATION
 HWOBJ=COM,NAME=C,VAR=B60:3,PAR="NAM=COM
       ADD=$2350 BD=9600 PAR=E BIT=8";
```

Objekt definuje komunikační kanál pro vysílání a příjem zprávy. Je definováno jméno objektu NAME a adresa řídících proměnných. Parametr PAR je typu string a udává parametry samotného komunikačního kanálu. Jeho popis je závislý na typu použitého protokolu. Podrobnější popis jednotlivých protokolů najdete v kapitole 9.4. Jedním z řídících registrů je registr DNODE, do kterého je třeba u vyšších síťových protokolů doplnit adresu adresáta kterému je právě vysílaná zpráva určena, do registru RSNODE se naopak automaticky zapíše adresa odesílatele právě doručené zprávy.

Tabulka předdefinovaných symbolických jmen:

Číslo bytu	*číslo registru v příkladu*	*symbolický název registru v příkladu*	*Význam*
Base+0	B60.0	C_OFF	komunikační kanál je ve stavu „off“
Base+0	B60.1	C_ON	komunikační kanál je ve stavu „on“
Base+0	B60.2	C_ERR	při procesu ON/OFF došlo k chybě
Base+0	B60.3		rezerva
Base+0	B60.4	C_SEND	komunikační kanál vysílá zprávu
Base+0	B60.5	C_SENDERR	poslední vysílání skončilo chybou
Base+0	B60.6	C_REC	v přijímacím bufferu je přijata nová zpráva
Base+0	B60.7	C_RECERR	poslední příjem skončil chybou
Base+1	B61	C_DNODE	adresa, na kterou se zpráva posílá – využívá se při vyšších protokolech
Base+2	B62	C_RSNODE	adresa, ze které přišla poslední platná zpráva – využívá se při vyšších protokolech

9.3 Procedury

Pro činnost zapnutí a vypnutí komunikace jsou definovány procedury **COMON(jméno)** a **COMOFF(jméno)**. Tyto procedury slouží k otevření, resp. ke spuštění obsluhy, a uzavření, resp. vypnutí obsluhy, komunikační linky. Vypnutí kanálu je signalizováno nastavením bitu **C_OFF** na 1. Zapnutí kanálu nastavením bitu **C_ON** na 1. Obě procedury pouze odstartují proces otevírání, resp. zavírání, komunikačního kanálu. Vlastní obsluha je prováděna paralelně se zpracováním kódu. Z tohoto důvodu může dojít k nastavení signalizačních bitů se zpožděním, které může být až 10s, při komunikaci přes modemy i déle.

Při procesu otevírání dojde k nastavení bitu **C_OFF** na 0, úspěšné provedení otevření kanálu je signalizováno nastavením bitu **C_ON** na 1, neúspěšné opětovným nastavením bitu **C_OFF** na 1 a nastavením bitu **C_ERR** na 1.

Při procesu zavírání dojde k nastavení bitu **C_ON** na 0, ukončení uzavření kanálu je signalizováno nastavením bitu **C_OFF** na 1. Pokud dojde při uzavírání k chybě, je nastaven bit **C_ERR** na 1.

Proceduru **COMON** můžeme doplnit o parametrizační string otevíraného komunikačního kanálu **COMON(jméno,par_str)**. Parametrizační string **par_str** se přidává před parametrizační string **par** zadaný při konfiguraci komunikační linky v sekci CONFIGURATION. Tento postup nám umožňuje vybrané parametry pro otevření komunikačního kanálu (např. bd rychlost) stanovit v uživatelském programu těsně před otevřením komunikačního kanálu, resp. v průběhu uživatelského programu vybrané parametry komunikace změnit zavoláním procedur **COMOFF** a **COMON** s novou hodnotou **par_str**.

Pokud chceme zprávu odvysílat, připravíme si ji do bufferu v paměti uživatelských registrů ve formátu:

Buffer	*Byte 0*	*Byte 1*	*Byte 2*	*...*	*Byte N*
Zpráva	Délka N	Data1	Data2	…	DataN

Zavoláním procedury **COMSEND(jméno,AddrBuffer)** s parametrem adresy bufferu danou zprávu odešleme, přesněji řečeno, zapneme automat, který na pozadí systému zprávu odešle. Kompletní odeslání zprávy je signalizováno nastavením bitu **C_SEND** na 0 (během vysílání 1), pokud skončíme s chybou, nastaví se příznak **C_SENDERR**.

Při použití vyšších síťových protokolů je potřeba definovat **dnode** adresáta. To se provede nastavením registru **C_DNODE** na požadovanou hodnotu. Obdobně při příjmu zprávy si můžeme přečíst, od koho nám zpráva přišla, přečtením registru **C_RSNODE**.

Pokud nechceme žádná data vysílat, automat neustále testuje, zda nebyla data přijata, pokud jsou data příjata, je nastaven bit **C_REC**. Nyní lze zavoláním procedury **COMRECEIVE(jméno,AddrBuffer,MaxLen)** přijatá data přesunout do námi připraveného bufferu AddrBuffer. Parametr MaxLen udává maximální délku přijaté zprávy včetně prvního Byte, který obsahuje skutečnou délku přijaté zprávy. Pokud je přijatá zpráva větší, přesune se pouze povolená část, zbytek se zahodí a nastaví se chyba při příjmu **C_RECERR** na 1.

Konkrétní použití komunikace je uvedeno v demonstračních příkladech.

Příklady: DCOM, DCOMPRT

9.4 Parametry komunikačních protokolů

9.4.1 Úvod

Tato kapitola obsahuje pouze přehledový popis konfiguračních parametrů PAR jednotlivých typů protokolů uváděného v sekci CONFIGURATION, resp. při volání procedury COMON. Pro podrobnější informace je třeba se s protokolem blíže seznámit a přečíst úplnou dokumentaci ke konkrétnímu komunikačnímu protokolu.

Jednotlivé protokoly jsou vždy zřetězeny a začínají od nejvyššího po nejnižší. U každého protokolu je uveden příklad a dále pak výpis všech parametrů s možností jejich hodnoty. Pokud některý vyšší protokol obsahuje pod sebou již definovány nižší protokol, není tato sekce dále komentována, je třeba si podle parametru NAM najít pro vysvětlení správný protokol.

Pro nejrychlejší a nejjednodušší porozumění jednotlivým protokolům vycházejte prosím vždy z dodaných demo příkladů. V případě nejasností se obracejte na dodavatele řídícího systému.

9.4.2 COM

Protokol slouzí pro přenos jednotlivých znaku.

```
"Name=COM COM=1 IRQ=4 ADD=$3F8 PAR=E BD=4800 LRB=1000";
```

Parametr	*Hodnota*	*Impilcitní*	*Význam*
NAM	COM	nutno nastavit	Parametr určující jméno komunikačního objektu, kterému jsou následující parametry určeny. Parametr musí být určen vždy a musí být uveden jako první.
COM	1|2|3|4	1	Parametr určující číslo sériového kanálu COM. aaa může nabývat hodnot 1 až 8. Adresy odpovídající jednotlivým kanálům COM jsou uvedeny v jednotce ChnTypes.
ADD	$addr	$3F8	Parametr určující adresu sériového kanálu COM. Použije se jen v případě, kdy komunikační adaptér i8250 není na standardní vstupně/výstupní adrese. $addr může nabývat hodnot 0 až $ffff.
IRQ	iii	4	Parametr určující číslo přerušení IRQ, na kterém adaptér i8250 žádá o zpracování přerušení. iii může nabývat hodnot 0 až 7.
BD	bbb	9600	Parametr určující přenosovou rychlost požadované sériové komunikace. bbb může nabývat hodnot 25 až 115200 Bd.
BIT	ddd	8	Parametr určující počet datových bitů v přenášeném znaku. ddd může nabývat hodnot 5 až 8.
PAR	O|E|N	O	Parametr určující paritu přenášených znaků. Hodnota O pro lichou paritu, E pro sudou paritu a N pro znak bez parity.
STO	1|2	1	Parametr určující počet stop-bitů v přenášeném znaku.
LRB	lll	nutno nastavit	Parametr určující velikost vstupního kruhového vyrovnávacího bufferu. Doporučujeme velikost bufferu volit jako pěti až deseti násobek nejdelší přenášené zprávy.

symbol ,\|' odděluje jednotlivé možnosti v hodnotě parametru.

9.4.3 COMBR

Protokol slouží obdobně jako protokol COM pro přenos jednotlivých znaků, pouze je přidán parametr BRK, který umožňuje zapínat, případně vypínat vysílání Break Interruptu na začátku zprávy.

```
"NAM=COMBR PAR=E BD=4800 LRB=1000 BRK=2";
```

Parametr	*Hodnota*	*Impicitní*	*Význam*
NAM	*COMBR*	*nutno nastavit*	*Parametr určující jméno komunikačního objektu, kterému jsou následující parametry určeny. Parametr musí být určen vždy a musí být uveden jako první.*
BRK	*aaa*	*5*	*Parametr nastavuje pauzu ve znacích pro Break Interrupt. aaa může nabývat hodnot 0 až 255. Pokud je hodnota aaa rovna 0, nebude se Break Interrupt vysílat.*

9.4.4 COMPB

Protokol slouží obdobně jako protokol COM pro přenos jednotlivých znaků, pouze je přidán parametr PB, který umožňuje zapínat, případně vypínat používání paritního bitu pro informaci o přenášení adresy.

```
"NAM=COMPB PAR=E BD=4800 LRB=1000 PB=ON";
```

Parametr	*Hodnota*	*Implicitní*	*Význam*
NAM	COMPB	nutno	Parametr určující jméno komunikačního objektu, kterému jsou následující

		nastavit	parametry určeny. Parametr musí být určen vždy a musí být uveden jako první.
PB	ON|OFF	ON	Parametr nastavující příznak používání nastavitelného paritního bitu při vysílání.

symbol ,| ' odděluje jednotlivé možnosti v hodnotě parametru.

9.4.5 PRT

Protokol slouží pro přenos ucelených zpráv s adresou odesilatele a příjemce. Adresu příjemce zprávy DNODE je třeba nastavit přímo v programu před prvním odesláním libovolné zprávy

```
"Name=PRT LSB=1000 NOD=20 Name=COM PAR=E BD=4800 LRB=1000";
```

Parametr	*Hodnota*	*Implicitní*	*Význam*
NAM	PRT	nutno nastavit	Parametr určující jméno komunikačního objektu, kterému jsou následující parametry určeny. Parametr musí být určen vždy a musí být uveden jako první.
NOD	aaa	0	Parametr definuje NODE stanice na komunikační síti. aaa může nabývat hodnot 0 až 255.
LSB	lll	nutno nastavit	Parametr určuje velikost bufferu, který se vyhradí pro vysílanou zprávu. Do tohoto bufferu je transformována vysílaná zpráva, která je předána jednotkám nižších komunikačních vrstev k odeslání. lll může nabývat hodnot 10 až 32000 byte.

9.4.6 TECOM

Protokol slouží pro přenos ucelených zpráv s automaty TECO. Jelikož se jedná o síťový protokol, je třeba před odesláním první zprávy nastavit v programu adresu příjemce (DNODE).

```
"NAM=TECOM MAS=MASTER NOD=100 LSB=200 NAM=COM ADD=$2320 IRQ=4 BD=9600 BIT=8 STO=2 PAR=E
LRB=500";
```

Parametr	*Hodnota*	*Implicitní*	*Význam*
NAM	TECOM	nutno nastavit	Parametr určující jméno komunikačního objektu, kterému jsou následující parametry určeny. Parametr musí být určen vždy a musí být uveden jako první.
MAS	MASTER | SLAVE	MASTER	Parametr nastavuje zařízení do módu MASTER nebo SLAVE, implicitně je v módu MASTER
NOD	Aaa	0	Parametr definuje NODE stanice na komunikační síti. aaa může nabývat hodnot 0 až 255.
LSB	Lll	nutno nastavit	Parametr určuje velikost bufferu, který se vyhradí pro vysílanou zprávu. Do tohoto bufferu je transformována vysílaná zpráva, která je předána jednotkám nižších komunikačních vrstev k odeslání. lll může nabývat hodnot 10 až 32000 byte.

symbol ,| ' odděluje jednotlivé možnosti v hodnotě parametru.

9.4.7 SAIA

Protokol slouží pro přenos ucelených zpráv s automaty SAIA. Jelikož se jedná o síťový protokol, je třeba před odesláním první zprávy nastavit v programu adresu příjemce (DNODE).

```
"NAM=SBUS MAS=MASTER NOD=100 LSB=200 DAT=ON NAM=COMPB ADD=$2320 IRQ=4 BD=9600 BIT=8 STOP=2
LRB=1000 PB=ON";
```

Parametr	*Hodnota*	*Implicitní*	*Význam*
NAM	SBUS	nutno nastavit	Parametr určující jméno komunikačního objektu, kterému jsou následující parametry určeny. Parametr musí být určen vždy a musí být uveden jako první.
MAS	MASTER | SLAVE	MASTER	Parametr nastavuje zařízení do módu MASTER nebo SLAVE
DAT	ON|OFF	OFF	Parametr nastavuje příznak používání datového režimu protokolu (On - používat datový režim, Off – nepoužívat datový režim).
NOD	aaa	0	Parametr definuje NODE stanice na komunikační síti. aaa může nabývat hodnot 0 až 255.
LSB	lll	nutno nastavit	Parametr určuje velikost bufferu, který se vyhradí pro vysílanou zprávu. Do tohoto bufferu je transformována vysílaná zpráva, která je předána jednotkám nižších komunikačních vrstev k odeslání. lll může nabývat hodnot 1 až 32750 byte.

symbol ,| ' odděluje jednotlivé možnosti v hodnotě parametru.

9.4.8 LECOM – připravuje se

není zatím plně zdokumentován

9.4.9 Rockwell Automation (Allen-Bradley)

Protokol slouží pro přenos ucelených zpráv s automaty AB pod protokolem DF1. Jelikož se jedná o síťový protokol, je třeba před odesláním první zprávy nastavit v programu adresu příjemce (DNODE).

```
"NAM=DF1 MAS=MASTER NOD=100 LSB=200 DAT=ON NAM=COM ADD=$2320 IRQ=4 BD=9600 BIT=8 STOP=2
LRB=1000";
```

Parametr	*Hodnota*	*Implicitní*	*Význam*
NAM	DF1	nutno nastavit	Parametr určující jméno komunikačního objektu, kterému jsou následující parametry určeny. Parametr musí být určen vždy a musí být uveden jako první.
MAS	MASTER | SLAVE	MASTER	Parametr nastavuje zařízení do módu MASTER nebo SLAVE
FHD	FULL |HALF	FULL	Parametr nastavuje příznak používání full-duplex nebo half-duplex režimu protokolu.
CRC	ON|OFF	ON	Parametr nastavuje příznak používání šestnáctibitového kontrolního součtu cyklickým polynomem (ON) nebo osmibitový kontrolní součet (OFF)
NOD	Aaa	0	Parametr definuje NODE stanice na komunikační síti. aaa může nabývat hodnot 0 až 255.
LSB	Lll	nutno nastavit	Parametr určuje velikost bufferu, který se vyhradí pro vysílanou zprávu. Do tohoto bufferu je transformována vysílaná zpráva, která je předána jednotkám nižších komunikačních vrstev k odeslání. lll může nabývat hodnot 1 až 32750 byte.

symbol ‚|' odděluje jednotlivé možnosti v hodnotě parametru.

10. Speciální SW objekty

10.1 Úvod

V této kapitole jsou popsány speciální SW objekty, umožňující zařadit do programu některou z dále popsaných funkcí.

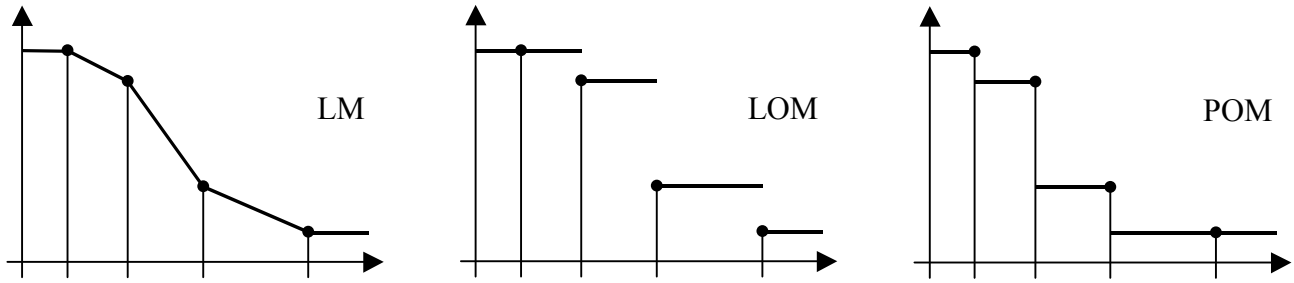
Použití jednotlivých objektů se deklaruje jejich uvedením v konfigurační sekci CONFIGURATION programu.

V následujících kapitolách jsou popsány konfigurační řádky včetně dalších funkcí a procedur nad těmito objekty pracujících.

10.2 SW-objekt TAB

```
CONFIGURATION
  SWOBJ=TAB,NAME=TAB1,X=BYTE:10,Y=BYTE:10;
```

Speciálním objektem projektu KB jsou tzv. tabulky. Každá tabulka má dva řádky, řádek X – proměnná X a řádek Y – proměnná Y a několik sloupců – počet udává proměnná Nb. Filosofie práce s tabulkami je následovná. Pomocí funkce SETTAB se nastaví obsah tabulky, nastavení obsahu lze provést i přímým přístupem na proměnné. Následně se provede seřazení jednotlivých sloupců v tabulce vzestupně podle hodnot uložených v řádku X. Procedura SETTAB provádí tento úkon automaticky, při přímém přístupu je pro řazení určena procedura SORTTAB. Poté je tabulka připravena pro další práci. Pomocí funkcí GETTAB a GETEKVTAB lze z tabulky získávat data. Funkce vlastně vrací hodnotu funkce, která je popsána množinou bodů obsažených v tabulce. Aproximace v mezilehlých bodech se provádí třemi základními metodami: lichoběžníková (LM), levá obdelníková (LOM) a pravá obdelníková (POM) metoda. Typ metody se zadává parametrem Formát. U všech aproximací platí následující pravidlo: pokud je hodnota před prvním prvkem X, automaticky se vrací hodnota prvního prvku Y, pokud je hodnota za posledním prvkem X, automaticky se vrací hodnota posledního prvku Y. Tímto postupem se předchází nutnosti limitace výstupu tabulky.



číslo bytu	*Číslo registru v příkladu*	*symbolický název registru v příkladu*	*Význam*
Base	automaticky	TAB1_NB	automaticky přidělená adresa registru s počtem prvků v tabulce
automaticky	automaticky	TAB1_X	adresa prvního prvku vektoru X
automaticky	automaticky	TAB1_Y	adresa prvního prvku vektoru Y

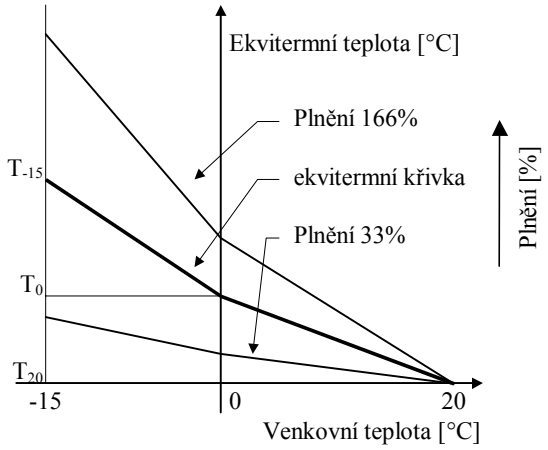
Jak již bylo řečeno, pro práci s tabulkou existuje několik speciálních funkcí, resp. procedur.

Procedura **SORTTAB(jméno tabulky)** srovná prvky tabulky podle hodnot prvků vektoru X.

Funkce **Y:=GETTAB(Jméno tabulky,Format,X);** vrátí z tabulky prvek Y odpovídající hodnotou vstupu X, parametr Format určuje typ aproximační metody.

Procedura **SETTAB(Jméno tabulky, [X1,Y1], [X2,Y2], [X3,Y3],…, [Xn,Yn])** provede nastavení tabulky na dané prvky. Po nastavení se automaticky zavolá funkce SORTTAB, při které dojde ke srovnání prvků tabulky.

Funkce **T:=GETEKVTAB(Jméno tabulky,X,Plnění);** je speciální případ funkce GETTAB s formátel LM, doplněné o vypočty pro práci s ekvitermní křivkou. Procedura má vstupní parametr plnění a podle jeho velikosti modifikuje vystupní hodnotu funkce (provede procentuální náklon křivky, rozsah parametru plnění může být od 0 do 255%). Na obrázku je ukázána modifikace ekvitermní křivky zadané třemi body v závislosti na parametru plnění.



10.3 SW-objekt SAVER

```
CONFIGURATION
 SWOBJ=SAVER,SAVE=[B5, B7, B10..B20,
       B100..B120];
```

Objekt definuje, které části uživatelských registrů se mají ukládat do chráněné sekce paměti - tzv. SAVE-sekce.

Při běhu programu se cyklicky (synchronně s prováděním procesu MAIN) kopírují data z vybraných uživatelských registrů do SAVE-sekce. Tím je zaručena možnost navázání na poslední konzistentní stav uživatelských proměnných v případě resetu systému např. po výpadku napájení resp. zhroucení běhu programu.

Po resetu systému se provádí tato posloupnost při inicializaci hodnot uživatelských registrů:

1. vynulují se všechny registry
2. inicializuje se hodnota registrů dle CONFIGURATION sekce programu
3. **hodnota registrů zapsaných do SAVE-sekce se obnoví dle údajů naposledy uchovaných.**
4. provede se inicializace, uvedená v proceduře INIT

Jako parametr objektu SAVER uvádíme parametr SAVE, za kterým následuje v hranatých závorkách vypsaná množina registrů, které se mají uchovat (délka se automaticky bere dle uvedeného typu registru). Můžeme použít i zápis pomocí intervalu - viz příklad.

Příklad: DSAVER

10.4 SW-objekt ARCHIV

```
CONFIGURATION
 SWOBJ=ARCHIVEFIRST, NAME=A1, SIZE=10, PER=10,
      SAVE=[B5,L6,B10..B20],LOAD=[B25,B30..B40];
 SWOBJ=ARCHIVELAST,NAME=A2,SIZE=100,PER=3,
      SAVE=[B5,I6,B10..B20],LOAD=[B100..B113],
      FORMAT=["NB  ","%4d",…]
```

Objekt slouží k uchovávání souboru hodnot. Hodnoty jsou uchovávány do archívu do jednotlivých záznamů (record), umístěných mimo uživatelskou datovou strukturu. Tyto záznamy je možno kdykoliv zpět načíst do uživatelských registrů. Archívů může být nadefinováno více, rozlišují se svým jménem. V každém z archívů se mohou ukládat rozdílné hodnoty v rozdílných režimech. Pro kompatibilitu s budoucími verzemi a lepší orientaci v archívu se doporučuje jako první položku rekordu ukládaného do archívu uvést proměnnou obsahující aktuální systémový čas (SYSPACKTIME).

Jsou zavedeny 3 druhy archívů:

1. Archív prvních uložených hodnot (ARCHIVEFIRST)
2. Archív posledních uložených hodnot (ARCHIVELAST)
3. Archív chybových hlášení (ARCHIVEERROR)

Archívy prvních uložených hodnot (ARCHIVEFIRST) a posledních uložených hodnot (ARCHIVELAST) zaznamenávají záznamy, složené z uživatelských registrů, uvedených v parametru **SAVE**. Ukládání záznamů se provádí buď periodicky s periodou uvedenou v parametru **PER** (v příkladu 10, resp. 3 sec) nebo, pokud parametr PER není uveden nebo má hodnotu 0, vždy po provedení standardní procedury **ARCHIVESAVE(jméno)**. Celkem se do archívu vejde maximálně tolik záznamů, kolik je uvedeno v parametru **SIZE**. Pokud je tento počet překročen, další záznamy se do archívu prvních uložených hodnot neukládají, jsou zapomínány, do archívu posledních uložených hodnot se nový záznam uloží místo nejstaršího, který je zapomenut.

Čteme-li tedy archív po delší době, než která je potřebná k jeho celkovému zaplnění, najdeme v archívu prvních, resp. posledních, SIZE záznamů. Celkový počet uložených záznamů v archívu je možno zjistit zavoláním standardní funkce **ARCHIVECOUNT (jméno)**, která vrátí skutečný počet uložených záznamů v archívu. Chceme-li archív smazat, můžeme použít buď standardní proceduru **ARCHIVECLEARALL(jméno),** která smaže všechny záznamy v archívu nebo standardní proceduru **ARCHIVECLEAROLD(jméno),** která smaže pouze 1 nejstarší záznam v archívu (výsledek ARCHIVECOUNT se zmenší o 1).

Archív chybových hlášení (ARCHIVEERROR) slouží k automatickému ukládání všech vzniklých "chybových" stavů:

- Okamžik nahrání nového uživatelského programu

- Okamžik resetu programu (po nahrání, po výpadku napájení nebo po zapracování ochrany watch-dog)

- Výskyt chyby při interpretaci p-kódu (snaha o dělení 0 apod.)

- Případný výskyt run-time chyby v kódu interpretu (systémová chyba výrobce SW)

Do tohoto archívu nelze uživatelsky zapisovat, nelze jej mazat, nelze určit maximální počet záznamů, lze z něj pouze číst.

Archív ARCHIVEERROR lze generovat maximálně jednou!

Všechny druhy archívů lze číst po jednotlivých záznamech explicitní standardní procedurou **ARCHIVELOAD(jméno,regbit).** Touto procedurou se do sady uživatelských registrů, nadefinované v parametru **LOAD** (pro množiny registrů v parametru SAVE a LOAD platí pouze to, že musí zabírat stejný počet bytů, předpokládá se však, že uživatel zvolí stejnou strukturu registrů a jejich typů) načte 1 záznam z archívu. Uživatelský registr typu bit **regbit** udává, zda se načtení záznamu povedlo (=1) nebo nikoliv (=0 - protože další záznam už neexistuje). Při prvním zavolání ARCHIVELOAD se čte první záznam archívu, a nastaví se ukazatel na něj, při každém dalším přečtení záznamu, se ukazatel posune o jeden záznam vpřed a ten se přečte.

Dále je implementována standardní procedura **ARCHIVELOADPREV(jméno,regbit).** Při vyvolání této procedury se ukazatel v archívu posune o jeden prvek zpět a tento prvek se přečte, výjimku tvoří první čtení z archívu, pokud ukazatel ukazuje na nepřečtený prvek, přečte se ten a ukazatel se nehýbe. Tímto postupem dochází ke čtení archívu pozpátku. Uživatelský registr typu bit **regbit** udává, zda se načtení záznamu povedlo (=1) nebo nikoliv (=0 - protože předchozí záznam už neexistuje).

Jiný záznam pro čtení můžeme nastavit zavoláním standardní procedury **ARCHIVESEEKFIRST(jméno)** (nastaví první záznam) nebo standardní procedury **ARCHIVESEEKLAST(jméno)** (nastaví poslední záznam).

U archívu typu ARCHIVEERROR musí mít LOAD parametr povinně 11 bytů s tímto významem:

```
SYMBOL
 Etime =L100; {longint-cas vyskytu chyby }
 EPrgVer=B104;{byte-verze programu}
 EPlace=B105; {byte-misto vyskytu chyby }
              {0=runtime,1=pcode,2=system}
 Ecode =B106; {byte-kod chyby }
 EASeg =W108; {word-adresa chyby=segment}
 EAOff =W110; {word-adresa chyby=offset}
CONFIGURATION
 SWOBJ=ARCHIVEERROR,NAME=AE,LOAD=[ETime,
       EPrgVer,EPlace,ECode,EASeg,EAOff];
```

Všechny archívy se ukládají do chráněného místa v paměti, a jejich obsah zůstává tudíž při výpadku napájení nebo resetu systému zachován.

Speciálními instrukcemi pro práci s archívy jsou procedury **ARCHIVELOADNB(jméno,ix)** – čtení položky o indexu **ix** z archívu jména **jméno** a procedura **ARCHIVESAVENB(jméno,ix)** s opačným významem. Tyto funkce mají dva parametry, jméno archívu a index prvku v archívu. Index je počítán od počátku vyhražené paměti počínaje číslem 0. Při tomto přístupu není brán zřetel na typ archívu (first, last), vždy se přečte položka o daném indexu. POZOR – data uložená v dané položce archívu nemusí být konzistentní, neboť se neprovádí kontrola, zda došlo k zápisu této položky. Zjednodušeně lze říci, že si z archívu lze udělat pole záznamů, ke kterým přistupujeme přes tyto dvě funkce.

Posledním parametrem, který můžeme zadat je parametr **FORMAT**, který umožňuje popsat vektor zálohovaných datových registrů uživatelským popisem, případně definovat formát výpisu v prohlížecím okně na straně PC.

Parametr **Formát** má vždy ke každé položce archívu dvě části, první část - znaky, které budou vypsány přímo a druhou část - řetězec znaků, který představuje specifikaci formátu pro zobrazení hodnoty dané položky.

Formátovací řetězec má následujícící syntaktický tvar:

"%" ["-"] [width] ["." prec] type

Řetězec musí začínat znakem "%"(procento). Po znaku "%" následují:

volitelný indicator zarovnání, ["-"]

volitelná specifikace šířky zobrazení, [width]

volitelná specifikace (např. počtu des. míst), ["." prec]

typ formátování argumentu type

Následující tabulka shrnuje možné hodnoty pro typ formátování argumentu:

type	*Význam*
d	Dekadické číslo. Argumentem musí být celočíselná hodnota (celočíselný registr). Hodnota je převedena na řetězec dekadických číslic. Jestliže formátovací řetězec obsahuje volitelnou specifikaci .prec , pak tato přikazuje, aby výsledný řetězec měl nejméně tolik číslic. Jestliže hodnota má méně číslic, pak je výsledný řetězec doplněn zleva nulami.
e	Vědecký formát (Scientific). Argumentem musí být regist typu Real. Hodnota je převedena na řetězec ve tvaru "-d.ddd...E+ddd". Výsledný retězec začíná znakem "-"(minus), je-li číslo záporné. Jedna číslice vždy předchází desetinné tečce. Celkový počet číslic ve výsledném řetězci (včetně té jedné před des. tečkou) je dán .prec specifikací ve formátovacím řetězci, pokud specifikace .prec není uvedena, použije se hodnota 15. Znak "E" exponentu ve výsledném řetězci je vždy následován znaky "+"(plus) nebo "-"(minus) a nejméně třemi dalšími číslicemi.
f	Číslo s pevnou řádovou tečkou (Fixed). Argumentem musí být regist typu Real. Hodnota je převedena na řetězec ve tvaru "-ddd.ddd...". Výsledný retězec začíná znakem "-"(minus), je-li číslo záporné. Počet číslic za desetinnou tečkou je dán .prec specifikací, pokud specifikace .prec není uvedena, použije se hodnota 2.
g	General. Argumentem musí být regist typu Real. Hodnota je převedena na nejkratší možný dekadický řetězec za použití formátu s pevnou des. tečkou nebo ve vědeckém formátu.
n	Číslo. Argumentem musí být regist typu Real. Hodnota je převedena na řetězec ve tvaru "-d,ddd,ddd.ddd...". Typ "n" formátu je obdobný typu "f" formátu s tím, že výsledný řetězec obsahuje oddělovače tisíců.
s	Řetězec znaků (String). Argumentem musí být znak nebo řetězec znaků. Řetězec nebo znak je vložen na místo specifikátoru "s". Pokud je uvedena specifikace .prec , pak značí maximální délku výsledného retězce. Jestliže by byl výsledný řetezec delší než toto maximum, pak je zkrácen.
x	Hexadecimální vyjádření čísla (Hexadecimal). Argumentem musí být celočíselná hodnota (celočíselný registr). Hodnota je převedena na řetězec hexadecimálních číslic. Jestliže formátovací retězec obsahuje .prec specifikaci, pak výsledný řetězec musí mít nejméně tolik číslic. Pokud by měl výsledný řetezec méně číslic, pak je doplněn zleva nulami na potřebný počet číslic.

Znaky určující typ formátování argumentu mohou být uvedeny buď jako malá nebo velká písmena, obě možnosti vedou ke stejnému výsledku.

Specifikace šířky zobrazení width určuje minimální šířku zobrazení hodnoty argumentu ve výsledném řetězci. Jestliže by byl výsledný retězec kratší než width, pak je doplněn mezerami. Implicitně jsou mezery doplněny před hodnotu, avšak je-li uveden indikátor zarovnání doleva, tj. znak "-" předcházející width, pak je výsledný řetězec doplněn mezerami zprava, tj. připojením mezer za hodnotu.

Příklady:

%5.2f - zobrazení real s dvěma des. místy

$%8.8x - zobrazení longintu HEX, např. $AB89CDF4

Příklad: DARCHIVE

10.5 SW-objekt PID regulátor

```
CONFIGURATION
 SWOBJ=PID, NAME=R, VAR=I100:28, TS=5;
```

Objekt definuje samostatný proces PID regulátoru, který dle nastavených parametrů samostatně zabezpečuje regulaci žádané měřené hodnoty.

Objekt pracuje nad 14 uživatelskými registry typu **integer** (v příkladu počínaje registrem I100, nepovinný parametr **VAR**, pokud není zadán, registry se přidělí automaticky), dále vystupuje pod jménem **R**, parametr **NAME.** Vzorkovací perioda **TS** je nastavena na 5 sec. Význam jednotlivých registrů je následující:

offset reg	*název par.*	*Význam*	*implicitní hodnota*	*symbolický název registru*
0	AUTO	flag autoregulace 0=U:=Uman 1=automatická regulace	1	R_AUTO
2	W	požadovaná hodnota vstupní veličiny (regulátor se snaží regulovat tak, aby W=Y)	0	R_W
4	Y	Naměřená hodnota, například vstupní kanál z A/D převodníku	0	R_Y
6	U	Spočtená výst. hodnota akčního zásahu (zpravidla v jiných jednotkách	0	R_U

		než W a Y)		
8	Uman	Požadovaná výstupní hodnota při přepnutí do režimu MANUAL (AUTO=0)	0	R_Uman
10	KP	proporcionální konstanta PID regulátoru	0	R_KP
12	KI	integrační konstanta PID regulátoru	0	R_KI
14	KD	derivační konstanta PID regulátoru	0	R_KD
16	RELK	Koeficient, kterým, pokud je regulační odchylka E=W-Y záporná, se násobí KP, KI a KD	1	R_RELK
18	MINU	Minimální požadovaná výstupní hodnota U	0	R_MINU
20	MAXU	Maximální požadovaná výstupní hodnota U	4095	R_MAXU
22	EGAP	Výpočet nové hodnoty regulátoru pro abs(E) větší EGAP	0	R_EGAP
24	IGAP	Je-li abs(E) větší IGAP, vypne se Integrační složka PID regulátoru	4095	R_IGAP
26	DGAP	Je-li abs(E) větší DGAP, vypne se Derivační složka PID regulátoru	4095	R_DGAP

Parametr **NAME=R** specifikuje název tohoto SW-objektu. Tento symbolický název objektu můžeme použít ve spojení se symbolickým názvem jednotlivých položek k symbolickému označení registrů I100 až I130 tak, jak je uvedeno v posledním sloupci předchozí tabulky.

Pokud při deklaraci objektu nepoužijeme parametr NAME, je takový objekt pojmenován stejně jako jeho typ, tj. v tomto případě PID. Vzhledem k tomu, že žádný identifikátor nesmí být v programu definován 2x, by se však hlásila chyba, kdybychom chtěli nadefinovat objekt PID vícekrát a nepoužili (vyjma jedné definice) parametr NAME.

Parametr **TS=5** udává periodu výpočtu (vzorkování) regulátoru v sekundách - zde se bude výpočet provádět každých 5 sec.

Příklad: DPID

10.6 SW-objekt PIDR regulátor

Pozn.: Tento objekt vzhledem k velké časové náročnosti výpočtu není dále rozvíjen

```
CONFIGURATION
 SWOBJ=PIDR, NAME=R, VAR=R100:16, TS=5;
```

Objekt definuje samostatný proces PID regulátoru, který dle nastavených parametrů samostatně zabezpečuje regulaci žádané měřené hodnoty.

Objekt pracuje nad 16 uživatelskými registry typu **real.** (počínaje registrem R100), dále vystupuje pod jménem **R.** Vzorkovací perioda TS je nastavena na 5 sec. Význam jednotlivých registrů je následující:

čís. reg	*název par.*	*Význam*	*implicitní hodnota*	*symbolický název registru*
0	AUTO	flag autoregulace 0=U:=Uman 1=automatická regulace	1	R_AUTO
1	W	požadovaná hodnota vstupní veličiny (regulátor se snaží regulovat tak, aby W=Y)	¨0	R_W
2	Y	Naměřená hodnota, například vstupní kanál z A/D převodníku	0	R_Y
3	U	Spočtená výst. hodnota akčního zásahu (zpravidla v jiných jednotkách než W a Y)	0	R_U
4	Uman	Požadovaná výstupní hodnota při přepnutí do režimu MANUAL (AUTO=0)	0	R_Uman
5	KP	proporcionální konstanta PID regulátoru	0	R_KP
6	KI	integrační konstanta PID regulátoru	0	R_KI
7	KD	derivační konstanta PID regulátoru	0	R_KD
8	RELK	Koeficient, kterým, pokud je regulační odchylka E W=Y E=---------- *100[%]- MAXY-MINY Záporná, se násobí KP, KI a KD	1	R_RELK
9	MINU	Minimální požadovaná výstupní hodnota U	0	R_MINU
10	MAXU	Maximální požadovaná výstupní hodnota U	100	R_MAXU
11	MINY	Minimální uvažovaná naměřená hodnota Y	0	R_MINY
12	MAXY	Maximální uvažovaná naměřená hodnota Y	100	R_MAXY
13	EGAP	Výpočet nové hodnoty regulátoru pro abs(E) větší EGAP	0%	R_EGAP
14	IGAP	Je-li abs(E) větší IGAP, vypne se Integrační složka PID regulátoru	100%	R_IGAP

15	DGAP	Je-li abs(E) větší DGAP, vypne se Derivační složka PID regulátoru	100%	R_DGAP

Parametr **NAME=R** specifikuje název tohoto SW-objektu. Tento symbolický název objektu můžeme použít ve spojení se symbolickým názvem jednotlivých položek k symbolickému označení registrů R100 až R115 tak, jak je uvedeno v posledním sloupci předchozí tabulky.

Pokud při deklaraci objektu nepoužijeme parametr NAME, je takový objekt pojmenován stejně jako jeho typ, tj. v tomto případě PIDR. Vzhledem k tomu, že žádný identifikátor nesmí být v programu definován 2x by se však hlásila chyba, kdybychom chtěli nadefinovat objekt PIDR vícekrát a nepoužili (vyjma jedné definice) parametr NAME.

Parametr **TS=5** udává periodu výpočtu (vzorkování) regulátoru v sekundách - zde se bude výpočet provádět každých 5 sec.

11. Seznam systémových registrů

symbolický název registru	*datový typ*	*Čtení/ Zápis*	*Význam*
SYSYEAR	W	Č	kalendářní rok 0-65535
SYSMONTH	B	Č	kalendářní měsíc 1-12
SYSDAY	B	Č	kalendářní den 1-31
SYSDOW	B	Č	den v týdnu 0=ne až 6=so
SYSHOUR	B	Č	hodiny 0-59
SYSMIN	B	Č	minuty 0-59
SYSSEC	B	Č	sekundy 0-59
SYSPACKTIME	D	Č	zapakovaný systémový čas
PROCVER	W	Č	aktuální verze procesoru
COMPVER	W	Č	aktuální verze překladače kterým byl přeložen nahraný program
PROGVER	B	Č	aktuální verze programu nastavená v sekci options
FastFreq	B	Č	aktuální nastavená rychlost vykonávání systémové procedury SYSFAST (10,20 nebo 50 ms)
MainTime	B	Č	Proměnná obsahuje průběžnou dobu provádění 1 průchodu systémovou procedurou sysmain v počtech mezitím provedených přerušení (počtu provedení systémové procedury SysFast, volaných zpravidla po 10 ms.) přečtením této proměnné ve dvou místech algoritmu a provedením odečtu můžeme získat informaci o časové náročnosti příslušné části tohoto algoritmu, pokud je takovýto algoritmus součástí kódu procedury Main nebo kódů obsluhy událostí (stisk kláves apod.)
MainMaxTime	B	Č/Z	Proměnná obsahuje maximální dobu provádění průchodu systémovou procedurou sysmain v počtech mezitím provedených přerušení (počtu provedení systémové procedury SysFast, volaných zpravidla po 10 ms.) V systémové proceduře SysMain se provádí čtení vstupů s režimem MAIN, tělo uživatelské procedury MAIN, zápis na výstupy s režimem MAIN a případná obsluha reakce na stisknutou klávesu, která je definována v těle popisu `terminal` v rámci popisu jednotlivých obrazovek. Přesahuje-li vrácené číslo hodnotu 100 při frekvenci SysFast 10ms, tj. celkovou dobu 1sec, je třeba pro bezchybnou funkci programu přepsat kód procedury MAIN, případně kód odezvy na stisk kláves tak, aby se prováděl při jednom jejím průchodu menší počet instrukcí, jinak hrozí zafungování hardwarové ochrany (watchdogu) a reset programu. Získané číslo je třeba porovnávat s potřebnou rychlostí reakce řídícího systému na události, detekované na vstupech, ovládaných v režimu MAIN. Je-li číslo příliš vysoké, může být odezva na vstupní signály příliš pomalá.
FastMaxTime	B	Č/Z	proměnná obsahuje maximální dobu provádění průchodu systémovou procedurou fast v procentech vzhledem k její maximální přípustné době provádění (zpravidla 10ms, viz však option FastFreq, kap. 6.17.2, str. 25) od posledního resetu systému resp. nahrání programu resp. programovému vymazání
FastOverFlow	V	Č/Z	Proměnná obsahuje hodnotu 0 nebo 1. Hodnota 1 (true) značí, že došlo k časovému přetečení při provádění systémové procedury Fast (proměnná FastMaxTime má hodnotu 100 a více). Ošetření tohoto stavu je nutno provést stejně jako je popsáno výše.
SysCom_State	V	Č	Proměnná obsahuje 1 pokud je obsluhována systémová komunikace, pokud komunikace neběží, obsahuje proměnná 0.
SysCom_Ini	V	Č/Z	Pokud chceme provést znovu inicializaci systémové komunikační linky, nastavíme hodnotu této proměnné na 1, jinak ji necháme v 0.
SysCom_Master	V	Č/Z	Při komunikaci s na systémové lince přes modem, můžeme při znovu-inicializaci nastavit, zda chceme sami vytáčet číslo 1, nebo čekat až nám někdo zavolá 0.

Do všech proměnných označených symbolem 'Z' můžeme v uživatelském programu volně zapisovat

12. Příklady

Pokud není uvedeno jinak, jsou všechny příklady určeny pro sestavu obsahující terminál řady TERM10.

12.1 Program DPRVNI

V této kapitole je vypsána zdrojová podoba programu DPRVNI.PRG, uvedeného na demonstrační disketě. Tento program slouží pro první seznámení s možnostmi zde popisovaného jazyka.

```
{---------------------------------------- }
{  KIT-Builder  Program                    }
{                                          }
{  (c) SofCon 1998                         }
{---------------------------------------- }
{Program slouzi pro zakladni seznameni se se strukturou programu v jazyku Kit-Basic}
{ --- Definice syst. konstant ------------ }
options
  PROGVER=1;           {verze tohoto programu}
{ --- Definice konstant ------------------- }
constant
  PRVNISMER=1;
  maxCITAC=2000;       {symbolicky nadefinovane konstanty}
{ --- Definice HW-objektu TERM10 ---------- }
configuration
  HWOBJ=TERM10, ADR=$2300, LED=$0F;
{ -- Definice symbolickych nazvu registru - }
symbol
  CASOVAC =word;       {casovac pro synchronizaci citani}
  CITAC  =integer;     {symbolické označení pro registr}
  SMER   =integer;     {symbolické označení pro registr}
  PERIODA =WORD;       {perioda pro urcovani rychlosti pocitani}
{ --- Definice jednotlivych obrazovek ----- }
{ Popis obrazovky c.0 terminalu TERM10 }
terminal TERM10:0;
begin
  font 0;                  {Nastavime font 0}
  position 10,10; print "Prvni program";  {Na pozici position vytiskneme text }
  font 1;                  {Nastavime font 1}
  position 20,30; print "CITAC=",CITAC:6;   {Vypis registru CITAC na obrazovku}
  position 20,40; print "SMER= ",SMER:6;    {Vypis registru SMER na obrazovku}
  position  20,50;  print  "PERIODA=",PERIODA:4,"  x10ms";  {Vypis  registru  PERIODA  na
obrazovku}
  position 20,100; print "Napoveda-stisknete F1";
  onkey  {definujeme reakci na stisk klavesy}
    '+':SMER:=1;  {po stisku klavesy "+", resp. "-" zmena hodnoty registru SMER}
    '-':SMER:=-1;
    'P':TERM10_SCRNO:=1;
    '0':CITAC:=0;
  end;
  help  {obrazovka napovedy, objevi se po stisku F1}
    position 10 ,10; print "NÁPOVĚDA";
    position 10 ,40; print "Citac pocita podle nastaveni";
    position 10 ,50; print "klavesami +/- nahoru nebo dolu.";
    position 10 ,60; print "Editace periody klavesou P.";
    position 10 ,70; print "Vynulovani klavesou 0.";
    position 10, 90; print "Konec na PC – AltX";
end;
{ ---------------------------------- }
{ Popis obrazovky c.1 terminalu TERM10 }
terminal TERM10:1;
begin
  font 0;
  position 10,10; print "Prvni program";
  font 1;
  position 20,30; print "CITAC=",CITAC:6;       {vypis registru CITAC na obrazovku}
  position 20,40; print "SMER= ",SMER:6;         {vypis registru SMER na obrazovku}
  position 20,50; print "PERIODA=     x10ms";
  font 0;  position 68,50;
  edit PERIODA:3,5,500; {editace hodnoty periody}
  font 1;
```

```
  position 20,100; print "Napoveda-stisknete F1";
  help      {obrazovka napovedy, objevi se po stisku F1}
    position 10 ,10; print "NÁPOVĚDA";
    position 10 ,40; print "Editace periody citani";
    position 10 ,50; print "Potvrdte klavesou ENTER";
    position 10, 90; print "Zruste klavesou ESC";
end;
{--- Definice procedury, která se provede 1x po RESETu ---}
procedure INIT;
begin
  SMER:=PRVNISMER;   {po RESETu ma SMĚR hodnotu +1}
  Perioda=100;       {nastaveni pocatecni hodnoty periody}
  TimerOn(Casovac,per10ms); {Spusteni casovace}
end;
{--- Definice hlavniho algoritmu, provadeneho cyklicky ---}
procedure MAIN;
begin
  if CASOVAC>=Perioda then{cekam az je napoctena hodnota periody a pak}
  begin
    Casovac:=Casovac-Perioda;                {snizeni casovace o periodu}
    TERM10_Led:=not TERM10_Led;                {inverze LED-diod na terminalu}
    CITAC:=CITAC+SMER;        {citani citace}
  end;
end;
{---------------- KONEC --------------}
```

12.2 Abecední seznam demo příkladů

Na instalační disketě je k dispozici celá řada dalších DEMO příkladů. Jednotlivé příklady slouží k demonstraci použití jednotlivých prvků KIT-BASICu. Některé příklady ukazují použití HW ovladačů a tudíž bez potřebného HW nebudou fungovat dle očekávání. V následujícím seznamu jsou příklady seřazeny abecedně s uvedením stránky výskytu bližšího popisu.

DARCHIVE 53
DBAR 33
DCOM 46
DCOMPRT 46
DCRC 20
DEDIT 30
DEDITP 30
DGONIO 19
DGONIOR 19
DGRAF 26, 28, 30, 31, 43
DGRAPH 32
DGRAPHXY 33
DHODINY 20
DHODINYG 20
DIOADDA 42
DIODIO01 39
DIODOO01 39
DIODXO01 39
DIOPBUS 39
DIOT10 40
DLEDKIT 22
DLPT 21
DONKEY 34
DPID 54
DPRVNI 57
DRND 20
DSAVER 51
DTERM01 27
DTEXT 26, 28, 29, 35, 43
DWAIT 34
START 4

13. Otázky a odpovědi

V této kapitole se setkáme s několika doporučeními a odpovědmi na nejčastěji kladené otázky. Přečtením této kapitoly si urychlíte cestu k vašemu konečnému řešení.

Omlouváme se, že některá doporučení budou velmi jednoduchá a prostá, ale z praxe se nám stále více potvrzuje, že nejčastější chyby se dělají v základních věcech, o kterých si většina lidí říká: „vždyť to je přece samozřejmé“, „toto jsem již třikrát zkontroloval“, „tady ta chyba být nemůže“, „to znám“ apod.

Jakou hodnotu má mít adresa HW objektu?

V programu KIT-Basicu se definují adresy HW objektů (terminál, vstupy a výstupy, komunikační linky atd.). V uživatelském manuálu, který dostáváte s konkrétními HW deskami je popsáno nastavení adresy desky, nejčastěji 300-360 hexadecimálně. Ve skutečnosti se většinou jedná pouze o offset skutečné přístupové adresy. Tuto adresu je pak třeba zvýšit o bázi, která je pro procesorovou desku KITV40 2000 hexadecimálně. Adresový prostor se pak obvykle přesouvá do oblasti 2300 až 2360 hexadecimálně.

Ve svém projektu používáme uživatelskou obsluhu komunikační linky. Program po spuštění hlásí chybu „*OpenCom*“. Jak tuto chybu odstranit?

Zkontrolujte správné nastavení konfiguračních parametrů komunikačního kanálu, nejčastěji bývají problémy s adresou portu.

Po spuštění programu KBDCON se komunikace rozeběhne, ale nejde číst a zapisovat program, fonty a bitmapy. Jak tuto chybu odstranit?

Nahrávání nového programu, fontů a bitmap do řídící jednotky je možné pouze při zastaveném běhu stávajícího aplikačního programu, zkontrolujte, zda je program zastavený.

Po zastavení a spuštění uživatelského programu prostřednictvím ovládacího programu KBDCON se nerozeběhne uživatelská komunikace, nebo TERM01. Jak tuto chybu odstranit?

Zkuste systém resetovat, pokud se vše v pořádku rozeběhne, není se dále čím zabývat, pokud komunikace, resp. TERM01, stále nejde, je třeba zkontrolovat nastavení parametrů komunikačního protokolu, resp. terminálu TERM01.

Nahraji řádně program, který mi nejde spustit.

S největší pravděpodobností je aktivní funkce test propojky PBUS při startu programu, bud umístěte propojku do správné polohy, 47-49=blokace, 47-48=povolení startu, nebo zapište na bitovou adresu 4067.0 hodnotu 0 – potlačení funkce testování propojky PBUS.