Programujeme řídicí systémy Kit s BP7 a o.s. Retos

Příručka uživatele a programátora

***SofCon*®spol. s r.o.**
Střešovická 49
162 00 Praha 6
tel/fax: +420 220 180 454
E-mail: sofcon@sofcon.cz
www: http://www.sofcon.cz

Verze dokumentu 1.40

Datum vydání: 21.06.2005
Datum posledního uložení dokumentu: 21.06.2005
(Datum vydání a posledního uložení dokumentu musí být stejné)

Obsah :

1. O dokumentu

1.1. Revize dokumentu

Verze dokumentu	Autor	Datum vydání	Popis změn
1.00	Web		- První vydání
1.10	Tum	19.05.2003	- Úprava dokumentu dle ISO9000
1.30	Wil	06.09.2004	- Doplnění dokumentu o popisy různých algoritmů a o způsobu používání firemních knihoven.
1.40	Wil	21.06.2005	- Přidán popis prostředí DOS188. - Zavedení používání verzí balíků knihoven v názvech adresářů LIB, LIBT, LIBV. - Přidáno stručné porovnání HW vybavení řídicích systémů Kit. - Rozšířen popis používání sériového rozhranní. - Přidán popis Ladicích výpisů na VGA.

1.2. Účel dokumentu

Tento dokument slouží jako příručka pro základní seznámení s vývojovými prostředky firmy ***SofCon® spol. s r.o*** pro tvorbu aplikací ve vývojovém prostředí Borland Pascal 7.0 s operačním systémem ReTOS pro řídicí systémy KitXxx firmy ***SofCon® spol. s r.o***.

1.3. Rozsah platnosti

Určen pro programátory a uživatele programového vybavení *SofCon*.

1.4. Související dokumenty

Tento dokument předpokládá znalost programování v prostředí Borland Pascal 7.0. Čtenář by se měl seznámit také s popisem operačního systému ReTOS (viz dokument [2]), pokud ho ve svých aplikacích používá.

[1] „Termíny a definice“ (Terminy.pdf)

[2] „o.s.Retos – Operační systém reálného časy“ (Retos.pdf)

[3] „CrtCom - Knihovny pro ladicí výpisy na CRT a vstup z klávesnice přes sériovou komunikaci“ (CrtCom.pdf)

2. Termíny a definice

Obecně používané termíny a definice jsou popsány v samostatném dokumentu [1].

3. Úvod

Tento dokument se snaží objasnit problémy, na které narazí programátor při tvorbě prvního programu pro řídicí systém KitV40, Kit386EXR nebo Kit188ER v šestnáctibitovém prostředí Borland Pascal 7.0. Podává první informace o obsahu balíku dodávaných knihoven firmou ***SofCon® spol. s r.o.***, o způsobech ladění, o možnostech jak nahrát aplikaci do řídicího systému a o způsobech programování.

4. Stručné porovnání HW vybavení řídicích systémů Kit

	KitV40	**Kit386EXR**	**Kit188ER**
Procesor	NEC V40	Intel 386EX	AMD Am188ER
Frekvence	16 MHz	33 MHz	standardně 49 MHz
Používaná instrukční sada	80286 (bez instrukcí pro DPMI) *		
Adresovatelný paměťový prostor	1MB		
Adresovatelný IO prostor	64KB		
RAM	volitelně do 512KB	volitelně do 1MB	1MB (napevno)
ROM	volitelně do 512KB FLASH/EPROM	volitelně do 1MB FLASH/EPROM	volitelně do 768KB FLASH
Rozdělení paměťového prostoru na RAM/ROM	RAM – spodních 512KB ROM – horních 512KB	RAM – spodních 512KB ROM – horních 512KB	RAM – spodních 256KB až 512KB ROM – horních 512KB až 768KB **
Možnost připojení VGA pro ladění	Ano přes EXPPC00	Ano přes sběrnici PC104	Ne ***
Možnost připojení klávesnice pro ladění	Ano přes EXPPC00	Ano přes modul PCKB	Ne ***
Velikost BIOS	8KB	12KB	12KB
Sběrnice	IOBus, PBus, Systém Bus (pro EXPPC00)	IOBus, PBus, PC104	IOBus
Komunikační porty	1 x USART V40 (i8251)	2 x standardní UART (i8250)	1 x UART Am188 2 x standardní UART (i8250)
Compact Flash	Ne		Ano
Řadič přerušení	standardní 8259	standardní 8259A	Am188
Časovače	8254 na IO adrese $003C	standardní 8254	Am188
WatchDog	Ano		
RTC	Ano		

* V prostředí Borland Pascal 7.0 můžete zvolit používání 286 instrukcí. V případě procesoru Kit386EXR můžete používat 386 instrukce například v částech aplikace psaných v TURBO ASM.

** Hranice mezi RAM a ROM (FLASH) je volitelná po 64KB, tj. od varianty 256KB RAM a 768KB FLASH po 512KB RAM a 512KB FLASH. Do překrývajících se prostorů je přístup možný pomocí speciálních funkcí.

*** Pro tyto účely byl však vytvořen balík knihoven CrtCom (jehož popis je uveden v dokumentu [3]), které umožňují výpisy na vzdálený CrtTerminál přes sériovou komunikaci. Tyto knihovny mají celou řadu výhod a jsou použitelné i na KitV40 a Kit386EXR.

Kit188ER je dále vybaven řadou signalizačních LED indikujících příjem a vysílání znaků na komunikačních portech. Navíc má k dispozici 2 uživatelské LED a 3 uživatelské zkratovací propojky. Nemá k dispozici sběrnici PBus, kterou lze ovšem v případě nutnosti nahradit za PBus na přídavné desce IOP nebo IOPCOM.

5. Vývojové prostředí Borland Pascal a systémové knihovny *SofCon*

Vývojové prostředí Borland Pascal se systémovými knihovnami LIB firmy ***SofCon® spol. s r.o*** (případně jejich nadstavbami LIBT a LIBV pro terminály a touch panely) je určené pro tvorbu a odladění složitých i jednodušších aplikačních programů, určených pro řídicí jednotky KITXxx, terminály TERMxx nebo pro kompaktní řídicí systémy KOMPAKTxx. Vlastní programy píše uživatel v programovacím jazyce Borland Pascal 7.0 (nutná znalost dynamických proměnných a alespoň základní znalost objektového programování), přičemž využívá bohaté nabídky procedur a funkcí, obsažených v systémových knihovnách LIB.

Překladač Borland Pascal 7.0 není součástí CD disku ***SofCon***, poněvadž není volně šiřitelný. Uživatel si jej musí koupit, například i ve firmě ***SofCon.***

Firma ***SofCon*** ve svých systémových knihovnách LIB nabízí řadu knihovních modulů, které slouží pro tvorbu aplikace a pro ladění programu. Knihovní jednotky jsou určeny pro překladač jazyka Borland Pascal 7.0. Jednotky se standardně dodávají ve formě .TPU. Uživatel dostává k dispozici soubory s "interface" sekcemi (tj. části zdrojových tvarů), manuály a příklady použití jednotlivých knihoven. Některé vybrané knihovny se dodávají v kompletní zdrojové formě a po konzultaci může uživatel získat i zdrojové formy některých dalších knihoven.

Knihovny obsahují ovladače ke všem složitějším periferním deskám, komunikační knihovny, operační systém reálného času RETOS, knihovny pro práci se systémovým časovačem Int08, knihovnu pro práci s flash pamětmi, regulační knihovny a další užitečné programy. Většina knihoven je napsaná objektově, ale naplno jsou objektové vlastnosti využity jen u knihoven pro terminály a u komunikačních knihoven.

5.1. Instalace knihoven na disk

Systémové knihovny LIB a knihovny pro terminály LIBT a LIBV firmy SofCon se standardně instalují na disk K do adresářů **K:\LIBvvvv**, **K:\LIBTvvvv** a **K:\LIBVvvvv**, kde **vvvv** je číslo verze daného knihovního balíku (např. LIB0510 je balík knihoven LIB verze 05.10). Disk K je virtuální disk, který se vytvoří například pomocí příkazu SUBST z příkazové řádky. Na tento disk se instaluje veškeré SW vybavení firmy SofCon a tedy i výše zmíněné knihovny LIB, LIBT, LIBV. Tím je dána uživateli jistá volnost umístit si toto SW vybavení na svůj libovolný disk a libovolnou cestu a poté tuto cestu příkazem SUBST nastavit na disk K:. Dříve se veškeré SW vybavení umísťovalo na disk C do kořenového adresáře, což bylo ale pro programátory příliš svazující. Na disk K je doporučeno umístit i prostředí Borland Pascal (tj. **K:\BP**\…).

5.2. Adresářová struktura knihoven

Systémové knihovny LIB se dodávají ve verzi pro prostředí **DOS7**, **TPP7**, **MCP7**, **AM188** a **DOS188**. Prostředí **DOS7** je standardní prostředí počítače PC v reálném režimu, kde lze aplikaci ladit se simulátory hardwarového prostředí, nebo s expanzí sběrnic PCKit a na ni připojenými periferními deskami. I prostředí **TPP7** je určeno pro počítač PC, ale v chráněném (DPMI – Dos Protected Mode Interface) režimu. Prostředí **MCP7** je určeno pro řídicí počítač KITV40 nebo KIT386. Prostředí **AM188** je určeno pro řídicí počítač KIT188 s aplikací spouštěnou z FLASH paměti a bez podpory DOSu. Prostředí **DOS188** je určeno pro řídicí počítač KIT188 s podporou DOSu a všech jeho diskových služeb.

Pouhým překladem aplikace s verzí stejných knihoven pro dané prostředí a drobnou úpravou aplikace (např. změnou adres periferií) přejdeme od PC (simulaci) do prostředí KIT (reálný hardware). Těmto prostředím odpovídají také adresáře, v nichž jsou jednotlivé knihovny přeloženy.

```
K:\LIBvvvv
      \DOS7
        *.tpu
        *.obj
        turbo.tpl (verze pro PC)
      \TPP7
        *.tpp
        *.obj
        tpp.tpl
      \MCP7
        *.tpu
        *.obj
        turbo.tpl (verze pro KitV40 a Kit386EXR)
      \AM188
        *.tpu
        *.obj
        turbo.tpl (verze pro Kit188ER)
      \DOS188
        *.tpu
        *.obj
        turbo.tpl (verze pro DOS Kit188ER)
```

Soubor TURBO.TPL (TPP.TPL pro DPMI) obsahuje přeložené knihovny jazyka Borland Pascal 7.0, které firma SofCon opravila a upravila pro své řídicí systémy. Proto používejte výhradně těchto .TPL souborů místo originálního TURBO.TPL a TPP.TPL dodávaného firmou Borland v instalačním balíku Borland Pascal 7.0. Konkrétní použití .TPL souborů bude uvedeno dále.

Interface sekce jednotlivých knihoven (soubory .INT) jsou umístěny do adresáře **INTF** (viz obrázek níže).

Kompletní zdrojové tvary vybraných knihoven (.PAS, .ASM, .INC a další soubory) jsou umístěny do adresáře **SRC**. V tomto adresáři jsou jednotlivé knihovny, případně knihovní balíky, umístěny do svých vlastních adresářů (viz obrázek níže).

Příklady použití jednotlivých knihoven jsou umístěny do adresáře **EXAMPLES** (viz obrázek níže).

```
K:\LIBvvvv
     \INTF
       *.int
     \SRC
       \_LIBVER
         LibVer.pas
         History.txt
       \CRC16
         Crc16.pas
         History.txt
       ...
       ...
     \EXAMPLES
       ...
```

Struktura knihoven LIBT a LIBV je stejná jako knihoven LIB. Tj. kompletní knihovny včetně Borland Pascalu budou mít následující strukturu:

K:	virtuální disk pro SW vybavení Pascal a *SofCon*
\LIBvvvv	systémové knihovny
\DOS7	přeložené knihovny pro PC v real mode
\TPP7	přeložené knihovny pro PC v DPMI
\MCP7	přeložené knihovny pro KitV40 a Kit386 v real mode
\AM188	přeložené knihovny pro Kit188 v real mode
\DOS188	přeložené knihovny pro Kit188 v real mode s podporou DOSu
\INTF	interface sekce zdrojových tvarů
\SRC	kompletní zdrojové tvary vybraných knihoven
\EXAMPLES	příklady
\LIBTvvvv	starší knihovny pro terminály
\DOS7	
\TPP7	
\MCP7	
\AM188	
\DOS188	
\INTF	
\SRC	
\EXAMPLES	
\LIBVvvvv	nové knihovny pro terminály a touch panely
\DOS7	
\TPP7	
\MCP7	
\AM188	
\DOS188	
\INTF	
\SRC	
\EXAMPLES	
\BP	Borland Pascal 7.0
\BGI	
\BIN	
...	

5.3. Verze knihoven firmy *SofCon*

Byla zavedena tato jednotná pravidla:

1. Každá samostatně použitelná knihovna má svoji verzi v konstantách **cVer** (textový formát) a **cVerNo** (zkrácený číselný BCD formát) definovaných v interface sekci této knihovny. Podrobný popis textového i číselného formátu je uveden v dokumentu ke knihovně **LibVer**, která rovněž definuje řadu konverzních funkcí z jednoho formátu do druhého. Tyto formáty verzí je doporučeno používat i pro značení verzí aplikačních programů.
2. Některé knihovny, které nelze použít samostatně, byly sloučeny to tzv. balíků knihoven (např. balík knihoven LdrLib). Každý takovýto balík má rovněž svoji verzi (konstanty cVer a cVerNo) v jedné ze svých unit.
3. Verze kompletních systémových knihoven **LIB** je totožná s verzí knihovny LibVer, tj. **LibVer.cVer** a **LibVer.cVerNo**. Dále je tato verze shodná s verzí uvedenou v názvu adresáře LIBvvvv.
4. Verze kompletních terminálových knihoven **LIBT** je totožná s verzí knihovny uATerm, tj. **uATerm.cVer** a **uATerm.cVerNo**. Dále je tato verze shodná s verzí uvedenou v názvu adresáře LIBTvvvv.
5. Verze kompletních knihoven pro terminály a touch panely **LIBV** je totožná s verzí knihovny IODrv, tj. **IODrv.cVer** a **IODrv.cVerNo**. Dále je tato verze shodná s verzí uvedenou v názvu adresáře LIBVvvvv.
6. Každá vytvořená aplikace by měla obsahovat svoji verzi ve výše uvedeném formátu. Tuto verzi by měla nějakým způsobem zobrazovat (na obrazovce terminálu, v komunikovaném parametru apod.). Dále by rovněž měla zobrazovat verzi knihoven, se kterými byla přeložena. Tj. používá-li pouze systémové knihovny LIB, zobrazovat LibVer.cVer. Používá-li systémových LIB i terminálových knihoven LIBT, měla by zobrazovat uATerm.cVer, protože terminálové knihovny jsou nadstavbou systémových, tj. verze systémových knihoven je z verze terminálových dohledatelná. Stejně tak platí pro knihovny pro terminály a touch panely LIBV.

Pozor! Při používání odkazů na konstanty cVer a cVerNo nezapomínejte používat i jméno příslušné knihovny a za tečkou teprve konstantu verze. Vyhnete se tak nejednoznačnosti při použití více knihoven v části USES.

Př.
```
Uses
  LibVer,
  HwSyst,
  Tick,
  ...;
Const
  cVer = '01.00,01.01.2010'; {verze aplikace}
Begin
  Write('Moje Aplikace: ',cVer);
  Write('Knihovny LIB : ',LibVer.cVer);
  Write('   HwSyst    : ',HwSyst.cVer);
```

```
Write('   Tick      : ',Tick.cVer);
...
```

6. Vytváříme první aplikaci

Programátor musí zvládnout nejen napsání vlastní aplikace, ale také překladač jazyka Pascal, připojit správnou verzi knihoven, vygenerovat BIN moduly pro procesor KIT a zavést je do něj. Všechny tyto akce jsou citlivé na správné nastavení podmínek překladu a cest k adresářům. Na demonstračním CD ***SofCon*** se nacházejí zdrojové soubory demonstračních programů, které jsou uloženy i s prostředím překladače a podmínkami překladu v souborech BP.TP a BP.DSK. Odtud se lze inspirovat a podmínky překladu, include soubory a konfigurační soubory kopírovat.

6.1. Umístění aplikací na Vašem disku

Firma ***SofCon*** doporučuje mít všechny aplikace uložené na disku **J:** a zde i vytvářet aplikace nové. Tento disk J je zpravidla virtuální, vytvořený například příkazem SUBST z příkazového řádku. Tj. ve skutečnosti může mít programátor své aplikace umístěné na libovolném disku a i na libovolné cestě, kterou poté nastaví jako virtuální disk J. Tím se usnadní zejména případný přenos aplikací mezi různými počítači a i mezi různými programátory. Překladač jazyka Pascal si totiž ukládá v konfiguračních souborech absolutní cesty na pracovní adresář, otevřené zdrojové texty apod. Pokud tedy aplikaci zkopírujete do jiného adresáře, musí se poté v prostředí Pascal nastavit správné aktuální cesty. Jednotné zavedení umísťování aplikací na virtuální disk J toto „otravné“ přenastavování cest nevyžaduje.

Příklad: Programátor Karel má aplikaci s názvem PROJECT1 a má ji fyzicky umístěnou na cestě D:\Karel\MyApp\PROJECT1. Tuto aplikaci si nahraje programátor Petr a uloží si ji na svém PC na cestu C:\MyPrj\PROJECT1. Pokud chce nyní aplikaci otevřít v prostředí Pascal, musí nyní nastavit absolutní cesty, což je poměrně „otravná“ operace. Řešení je jednoduché. Programátor Karel má nastaven virtuální disk J na svoji cestu s aplikacemi D:\Karel\MyApp\. Tj. pro svoji aplikaci používá cestu J:\PROJECT1. Pokud Petr si tuto aplikaci zkopíruje do svého adresáře (jako v předchozím případě na cestu C:\MyPrj\PROJECT1) a pokud i on má nastaven virtuální disk J na svoji cestu pro aplikace (J = C:\MyPrj\), nebude nyní jeho aplikace J:\PROJECT1 vyžadovat žádné přenastavování adresářů.

Pozn: Virtuálním diskem **J:** pro aplikace a virtuálním diskem **K:** pro systémové prostředky pro tvorbu aplikací firma ***SofCon*** zavedla standard, který jsou povinni dodržovat všichni firemní programátoři a který se rovněž doporučuje ostatním programátorům, kteří vývojové prostředky firmy *SofCon* používají. Písmena J a K byla volena tak, aby vyhovovala co nejširšímu okruhu uživatelů. Pokud bohužel některý programátor má již některý z těchto virtuálních disků vyhrazen pro své jiné účely, bude muset pro aplikace či systémové prostředky *SofCon* použít disk jiný a bude muset případné rozdílné cesty přenastavovat ručně.

6.2. Pracovní adresáře aplikace

V minulé kapitole jsme si ujasnili, že aplikace umísťujeme na disk J. Pokud budeme mít aplikaci s názvem My1stApp, vytvoříme tento adresář na disku J, tj. aplikace se bude nacházet na cestě **J:\My1stApp**.

Vývojové prostředky firmy *SofCon* umožňují, že programátor není omezen jen na překlad aplikace pro řídicí systém KIT, ale může provést překlad stejných zdrojových textů pro prostředí DOS na PC a tím do značné míry svoji aplikaci odladit. Přístupy k fyzickému hardware řídicího systému, jakým mohou být například rozšiřující desky digitálních vstupů a výstupů si může programátor nasimulovat například vstupem z klávesnice. Samozřejmě svoji aplikaci doplní o podmíněný překlad simulačních algoritmů (pokud čtenář nezná pojem "podmíněný překlad v jazyce Pascal", měl by si své znalosti v tomto ohledu doplnit, jelikož další popis tuto znalost předpokládá).

Př:
```
{ifdef VerMC}
  {čtení vstupu 1 z desky IODXO01 na bázové adrese $2300}
  if (port[$2300] and $01) <> 0 then ...
{$else}
  {simulace vstupu 1 z klávesnice klávesou „1"}
  if Keypressed then
    if Readkey='1' then ...
{$endif}
```

Poněvadž verze aplikace do prostředí DOS (prostředí počítače PC) a do prostředí KIT (prostředí procesoru Kit) potřebují různé knihovny firmy *SofCon*, různé základní knihovny Borland Pascalu (soubor turbo.tpl), různé nastavení a konfigurači překladače, jeví se jako nejjednodušší překládat aplikaci pro každé prostředí do jiného adresáře a pracovat se společnými zdrojovými texty .PAS. Podmínky překladu lze umístit do zvláštního include souboru (zpravidla SETS.INC), který musí být umístěn v pracovním adresáři pro každé prostředí, tj. nikoliv v adresáři se společnými zdrojovými texty.

Struktura Vaší aplikace bude vypadat zhruba následovně:

```
J:\My1stApp                 - adresář Vaší aplikace
    !Readme.txt             - stručné informace o Vaší aplikaci
    History.txt             - historie verzí (změn) Vaší aplikace
    *.PAS                   - zdrojové texty
    \VerPC                  - pracovní adresář pro prostředí DOS
      Sets.Inc              - nastavení direktiv pro podmíněný překlad
      BP.TP, BP.DSK         - konfigurace překladače Borland Pascal
      *.Tpu                 - přeložené zdrojové texty
      Turbo.tpl             - knihovny Pascal pro prostředí DOS
      My1stApp.Exe          - přeložená a spustitelná aplikace
    \VerTPP                 - pracovní adresář pro prostředí DPMI
      Sets.Inc
      BP.TP, BP.DSK
      *.Tpp                 - přeložené zdrojové texty
      Tpp.tpl               - knihovny Pascal pro prostředí DPMI
      My1stApp.Exe          - přeložená a spustitelná aplikace
    \VerMc                  - pracovní adresář pro prostředí KIT
      Sets.Inc
```

`BP.TP, BP.DSK`		
`*.Tpu`	-	přeložené zdrojové texty
`Turbo.tpl`	-	knihovny Pascal pro prostředí KIT
`My1stApp.Exe`	-	přeložená (ale na PC nespustitelná) aplikace
`My1stApp.Map`	-	detailní .MAP soubor, který je nezbytně nutný pro nahrání aplikace do systému KIT

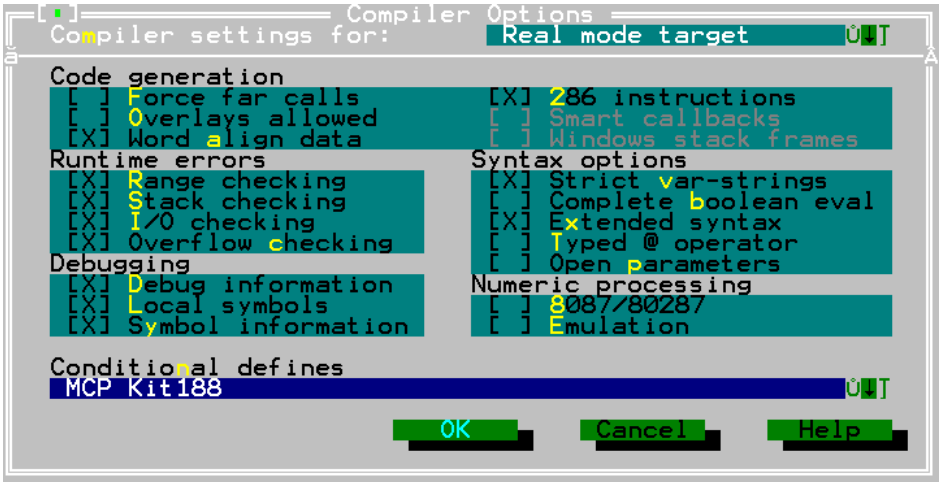
Soubory Turbo.tpl a Tpp.tpl nakopírujte do vašich pracovních adresářů ze systémových knihoven. Tj. pro prostředí DOS z K:\LIBvvvv\DOS7, pro Prostředí DPMI z K:\LIBvvvv\TPP7 a u prostředí KIT z K:\LIBvvvv\MCP7 nebo z K:\LIBvvvv\AM188 podle řídicího systému, který používáte. Pro KitV40 nebo Kit386EXR použijte Turbo.tpl z K:\LIBvvvv\MCP7 a pro Kit188ER z K:\LIBvvvv\AM188. Pokud náhodou použijete pro překlad nesprávný Turbo.tpl, bude Vám v lepším případě překladač hlásit chybu "Unit version mismash", v horším případě Vaše aplikace nemusí fungovat.

6.3. Překladač a prostředí Borland Pascal 7.0

Prostředí Pascalu můžete spustit několika způsoby. Buďto příkazem BP.EXE z příkazového řádku v příslušném pracovním adresáři pro dané prostředí (např. v J:\My1stApp\VerPC). Pro tento způsob je nutná nastavená cesta PATH k souboru BP.EXE. To je možné provést např. v Autoexec.Bat příkazem PATH=K:\BP\BIN;… Dalším způsobem je pomocí zástupce na ploše Windows. Tento zástupce spouští K:\BP\BIN\BP.EXE a jako pracovní adresář používá pracovní adresář pro dané prostředí vaší aplikace (např. J:\My1stApp\VerPC).

Nastavení překladače Pascalu obsahují soubory BP.TP a BP.DSK. Toto nastavení může být pro různá prostředí (DOS, KIT, atp.) odlišné.

1) V menu „File | Change dir …" nastavit cestu k aktivnímu pracovnímu adresáři. Tj. pro náš příklad J:\My1stApp\VerPC, J:\My1stApp\VerTPP nebo J:\My1stApp\VerMC podle právě používaného prostředí.
2) Pro rozsáhlejší projekty je vhodné nastavit v menu „Compile | Primary File" hlavní soubor aplikace.
3) V menu „Options | Compiler" nastavit:



Obrázek: příklad pro Kit188ER.

a) Cílový mód (Compiler settings for). Pro prostředí DPMI na PC zvolit „Protected mode target“, v ostatních případech „Real mode target“ (viz obrázek).

b) Nastavení překladu kódu (Code generation). Pro všechna prostředí lze použít 286 instrukce (i pro procesory KitV40 a Kit188ER). Doporučuje se použít „Word align data“. Pro jakékoliv prostředí KIT se nesmí použít „Overlays allowed“

c) Nastavení RunTime errors. Doporučuje se zapnout signalizaci všech chyb.

d) Nastavení Debugging. Doporučuje se zapnout kompletní debug informace.

e) Nastavení Syntax options závisí na uvážení konkrétního programátora.

f) Nastavení matematického koprocesoru (Numeric processing). Pro prostředí PC lze zapnout jakoukoliv volbu, ale pro prostředí KIT se nesmí použít žádná volba.

g) Nastavení podmínek překladu (Conditional defines). Pro prostředí DPMI na PC se používá standardní direktiva „DPMI“, kterou definuje překladač automaticky při volbě „Protected mode target“, tj. není ji nutno psát do Conditional defines. Pro všechna prostředí na PC se používá direktiva „DOS“. Pro všechna prostředí KIT se používá direktiva „MCP“. Pro procesor Kit188ER se krom direktivy „MCP“ používá dále direktiva „Kit188“. Programátor si může zvolit celou řadu dalších direktiv, které bude ve své aplikaci používat (např. DebCrt pro ladicí výpisy na VGA v txt módu). Tyto direktivy je zpravidla vhodné umísťovat do zvláštního souboru SETS.INC. Tento soubor poté používají VŠECHNY jednotky aplikace (příkaz {$I SETS.INC}).

4) V menu „Options | Memory sizes“ nastavit pro prostředí KIT velikost Stack size. Velikosti Low heap limit a High heap limit zde nemají vliv na velikost HEAPu. Tato velikost se nastavuje až v RetosDebuggeru resp. KitLoaderu (viz kapitola „7 Jak zavést aplikaci do procesoru Kit“).

5) V menu „Options | Linker“ nastavit pro prostředí KIT detailní MapFile s ukládáním na disk. Z tohoto .MAP souboru se berou v RTD informace, jak z EXE souboru vytvořit ROM moduly pro procesor Kit.

6) V menu „Options | Directories“ nastavit cesty ke zdrojovým souborům, knihovnám a do adresáře knihoven Pascalu. Doporučujeme používat tečkové konvence. Aktuální adresář je „.\;“, nadřazený adresář je „..\;“. Příklad pro verzi PC je na následujícím obrázku:

7) V menu „Options | Tools“ nastavit pro prostředí KIT cestu pro volání programu RetosDebugger (RTD.EXE) resp. KitLoader (KitLoader.EXE) a do Command line zadat: „$EXENAME /G”. Pozor! KitLoader.EXE má delší název než 8 písmen, proto s DOSovských programů jej je nutné volat jako KitLoa~1.EXE.

6.3.1. Překlad

Borland Pascal umí dva stupně překladu. První stupeň je Compile, kde se překládá pouze Primary File nebo aktivní soubor a soubory s uloženými změnami. Od ostatních zdrojových souborů se použijí již existující TPU soubory. Při rozsáhlejších změnách je vhodné zvolit druhý stupeň překladu Build. Při něm se znovu vytvoří všechny TPU od dosažitelných zdrojových programů. I tento stupeň však "nahlíží" do existujících TPU souborů a například, po výměně knihoven za novější verzi, občas selže a hlásí "unit version mismatch". Potom nepomůže nic jiného, než všechny aplikační TPU soubory smazat (pozor ať si nesmažete TPU soubory knihoven v adresářích LIB apod.) a vše znovu přeložit.

7. Jak zavést aplikaci do procesoru Kit

Pro řídicí systémy **KitV40** a **Kit386EXR** lze aplikaci spouštět pouze z pamětí FLASH/EPROM či RAM (viz následující kapitola).

Pro řídicí systém **Kit188ER** existují tyto možnosti spouštění aplikace:

1. Program je uložen v binární podobě ve FLASH paměti (stejné jako u KitV40 nebo Kit386) a ve FLASH paměti je přímo spuštěn (viz následující kapitola).
2. Program je uložen v binární podobě ve FLASH paměti, po inicializaci procesorové jednotky je překopírován do paměti RAM a spuštěn. Jedná se o podobný způsob jako v předchozím bodě, jen s tím rozdílem, že v konfigurační tabulce BIOSu se musí zaškrtnout položka „Run in RAM“.
3. Program je uložen v binární podobě na Compact Flash kartě, po inicializaci procesorové jednotky je soubor nahrán do paměti RAM procesorové jednotky a spuštěn (viz kapitola „7.2 Umístění aplikace na Compact Flash kartě Kit188ER v binární podobě“).
4. Program je uložen v .EXE podobě na Compact Flash kartě s operačním systémem FreeDos. Pro spuštění aplikace pod OS FreeDos použijeme systémový soubor Autoexec.Bat (viz kapitola „7.3 Umístění aplikace na Compact Flash kartě Kit188ER v EXE podobě s OS FreeDOS“).

7.1. Nahrání a spuštění aplikace v paměti FLASH/EPROM

Máme-li správně nastaven překladač BP7 pro prostředí KIT, vytváří se po překladu do pracovního adresáře EXE soubor s programem a MAP soubor s popisem přiřazení adres. Nyní zavoláme z prostředí překladače BP7 (formou TOOLS) program RetosDebugger (zkráceně RTD) resp. jeho novější verzi pro Windows KitLoader. Nastavení Tools bylo popsáno výše. Pomocí RTD/KitLoader z EXE souboru aplikace

vytvoříme moduly pro ROM a RAM do paměti procesoru Kit. Jedná se o standardní ROM moduly Biosu PC, které Bios v paměti vyhledává a spouští. Program je tvořen jedním RAM modulem a minimálně jedním modulem ROM. Podrobný popis programu RTD je popsán v dokumentu „RTD“. Podrobný popis programu KitLoader je popsán v dokumentu „KitLoader“.

RAM modul obsahuje jednotky Data, Stack a Heap. Uspořádání těchto jednotek v paměti RAM je popsáno v kapitole „9.5 Paměť RAM“.

Jeden ROM modul může být dlouhý maximálně 128kB. Proto je potřeba větší aplikaci rozdělit do více ROM modulů (ROM1, ROM2 …). Moduly ROM obsahují kód programu. Jednotky (unit) mohou být do ROM modulů poskládány celkem libovolně s výjimkou jednotky --Loader--, která musí být umístěna v adresové oblasti C0000h až FE000h. Jednotka --Loader-- totiž obsahuje spouštěcí hlavičku celé aplikace. Bios po startu vyhledává spustitelné ROM moduly pouze v adresové oblasti počínaje adresou C0000h směrem nahoru. Najde-li platnou spouštěcí hlavičku aplikace, spustí ji. V případě, že nenalezne žádnou platnou aplikaci, spustí aplikaci Bios Monitor, která komunikuje po sériové lince s RTD/KitLoader a umožňuje nahrání řídicí aplikace a některé ladicí prostředky.

U řídicích systému KitV40 a Kit386EXR je v oblasti ROM od A0000h do C7FFFh umístěna Videoram a BIOS VGA karty, která lze využít pro ladění. Do této oblasti standardně nemůžeme umístit naši aplikaci. Řídicí systém Kit188ER vyhrazenou oblast pro VGA nemá a tudíž lze celou ROM oblast využít pro aplikaci.

V případě potřeby využití výše uvedené oblasti KitV40 nebo Kit386EXR pro aplikaci je třeba nepoužívat VGA kartu pro ladění, navíc je však třeba použít pro KitV40 jiný obsah GAL paměti na procesorové desce - příslušnou podporu poskytne firma SofCon. Pro Kit386EXR se musí v RTD/KitLoaderu patřičně nastavit konfigurační tabulka.

V oblasti FE000h až FFFFFh je umístěn BIOS na desce KITV40, v oblasti FD000h až FFFFFh je umístěn BIOS na desce KIT386EXR a KIT188ER. Do těchto oblastí též nelze umístit aplikaci (a ani to RTD/KitLoader nepovolí).

Do procesoru lze moduly aplikace zavést třemi způsoby:

a. Pomocí RTD/KitLoader, který umí komunikovat s Biosem procesoru Kit a ve spolupráci s ním umí automaticky zavádět moduly do paměti FLASH i RAM. Pomocí propojek na procesoru lze zrušit běžící aplikaci a nastartovat pouze Bios. Po navázání spojení s Biosem vyměníme kód aplikačního programu. Podrobnější postup je rovněž popsán v kapitole „9.6.1 Nahrání nové verze aplikace do paměti Flash“.

b. Všechny Rom moduly i Bios vypálit v programátoru pamětí na správných adresách do paměti Eprom nebo Flash. Tuto paměť zasunout do patřičného soklu na procesorové desce.

c. Všechny Rom moduly a Bios nahrát do simulátoru Eprom, který nahrazuje v procesoru paměť Eprom. Přímo z RTD je možné volat obslužný program simulátoru. Pro simulátor SimEprom 02 firmy Elnec existuje ve firmě ***SofCon*** další programová podpora.

Programy RetosDebugger i KitLoader mají samostatný manuál, ve kterých jsou způsoby tvorby ROM modulů a jejich zavádění popsány.

7.2. Umístění aplikace na Compact Flash kartě Kit188ER v binární podobě

Při spouštění aplikace z Compact Flash karty je nutno nejprve CF kartu patřičně připravit (zformátovat): CF kartu vložíme do Kit188ER. Do paměti FLASH nahrajeme pomocí programu KitLoader aplikaci FDISK188. Tato aplikace zkontroluje MBR a BootSector, pokud MBR chybí, bude vytvořen (vytvoří se jedna primární partition o maximální velikosti CF). Po skončení programu vložíme kartu do čtečky CF karet PC. Pokud se kartu nepovede otevřít, musíme jí zformátovat (tento stav nastane pokud neexistoval MBR sektor, nebo byl poškozen). Po těchto úpravách můžeme na CF vytvořit potřebné soubory. Pak vrátíme CF zpět do Kit188ER a opět spustíme aplikaci FDISK188 (z důvodu korekce BootSektoru, který byl s největší pravděpodobností formátem přepsán).

Pozn: Pro účely knihovny CFlash (obdoba knihovny ATMFlash) je vhodné hned po zformátování vytvořit soubor pro odkládání dat na CF kartu v lineárním módu bez použití operačního systému FreeDos. Soubor může mít libovolnou velikost s ohledem na potřeby vaší aplikace. Soubor pro odkládání dat na CF kartu v lineárním módu je nutné vytvořit hned po zformátování proto, aby celý soubor byl na disku uložen lineárně, tj. nebyl fragmentován.

Při spouštění aplikace z Compact Flash karty je celý adresní prostor (1MB) procesorové jednotky pamětí typu readwrite RAM. To znamená, že žádná z částí paměti není ošetřena proti neoprávněnému zápisu, jako je tomu například v jiných případech při zápisu do horní oblasti adresního prostoru, který přistupuje k paměti FLASH.

V konfigurační tabulce KitLoader nastavíme příznak „Run from CF“ a do okénka „Boot file name“ zadáme název BIN souboru aplikace včetně přípony. Tento soubor vygenerujeme pomocí programu KitLoader stejným způsobem jako při spouštění z paměti FLASH, tj. výsledkem je jeden nebo více ROM modulů. V případě že se celá aplikace vejde do 128KB (tj. výsledkem je jeden ROM modul), zadáme do okénka „Segment“ konfigurační tabulky hodnotu start segmentu tohoto modulu zadaném v aplikaci KitLoader v okně „Memory Design“. V případě, že je aplikace větší než 128KB a nevejde se tudíž do jednoho ROM modulu, musíme pomocí speciální funkce aplikace KitLoader „sešít“ jednotlivé ROM moduly do jednoho výsledného BIN souboru. Do okénka „Segment“ konfigurační tabulky zadáme hodnotu start segmentu toho modulu, který je adresně nejníže. **Tento modul musí rovněž obsahovat hlavičku zavaděče --LOADER--.** Tím máme vytvořen BIN soubor aplikace, který můžeme nahrát na PC běžným způsobem (např. pomocí čtečky CF karet) do kořenového adresáře CF karty.

Po spuštění procesorové jednotky bude binární soubor aplikace nahrán z CF karty do paměti RAM na adresu zadanou v konfigurační tabulce a spuštěn.

7.3. Umístění aplikace na Compact Flash kartě Kit188ER v EXE podobě s OS FreeDOS

Compact Flash kartu pro potřeby operačního systému FreeDos připravíme stejným způsobem popsaným na začátku předchozí kapitoly. Na CF kartu nahrajeme firmou SofCon upravené jádro operačního systému FreeDos (*kernel.sys*), příkazový interpret (*command.com*) a dávkový soubor *autoexec.bat*, který je zpracováván při startu systému. Tyto soubory musí být umístěny v kořenovém adresáři CF karty. Do souboru *autoexec.bat* uvedeme jméno spouštěné aplikace (stejná syntaxe jako u „klasického autoexecu“ v systému DOS).

Procesorovou jednotku Kit188ER zkonfigurujem pomocí aplikace KitLoader způsobem popsaným v předchozí kapitole, přičemž obsah okénka „Boot file name“ změníme na *kernel.sys*, okénko „Segment“ nastavíme na hodnotu $100. Propojku na Kit188ER pro volbu Apl/BMon zkratujem.

Pozn: Po resetu bude Kit188ER nejprve hledat aplikaci na Compact Flash kartě (tj. kernel.sys), pokud selže (ve slotu nebude Compact Flash) bude hledat aplikaci v paměti FLASH.

Více informací o provozování aplikací pod OS FreeDos lze nalézt v dokumentu „Začínáme s OS FreeDos na Kit188ER“.

8. Možnosti ladění programu

Jednou ze silných stránek šestnáctibitového softwarového prostředí ***SofCon*** je možnost pohodlného ladění aplikace ve vývojovém prostředí Borland Pascal.

a. Aplikaci lze ladit v první fázi ve verzi DOS v počítači PC a místo hardware napsat simulátory. Ty simulují funkci hardware a mohou také periodicky vypisovat obsah zajímavých proměnných na monitor. V tomto stupni se dá aplikace velmi dobře vytvořit a zhruba odladit v integrovaném prostředí Borland Pascal 7.0, nebo pod TurboDebugerem. Nepotřebujeme k tomu nic jiného než počítač PC.

Když už potřebujeme přistupovat na reálný hardware, máme několik možností ladění:

b. Můžeme zůstat v režimu překladu DOS na počítači PC a počítač doplnit kartou PCKit na ISA sběrnici nebo LptKit přes paralelní port. Na kartě je PBus i IOBus, pouze se nacházejí na jiných adresách. K desce PCKit/LptKit připojíme ostatní hardware a z počítače PC vlastně uděláme procesorovou desku Kit. Můžeme ladit v integrovaném prostředí Borland Pascal i pod TurboDebugerem jako v předchozím případě. Rozdíl je v rychlosti počítačů, tj. v rychlosti vykonávání výpočtů apod. Dále musíme nastavit jiné adresy I/O prostoru, jiné vektory přerušení a nemůžeme přistupovat na absolutní adresy paměti RAM a FLASH jako v KIT. Změny se snadno realizují podmíněným překladem. Tento režim poskytuje nejmocnější ladicí možnosti reálné aplikace. Takto se dají odladit aplikace řízení celých velkých strojů.

c. Přejdeme s aplikací do reálného počítače KitV40 nebo Kit386, ke kterému připojíme kartu VGA. Na VGA monitor budeme vypisovat ladicí výpisy, jako jsou například obsahy proměnných, které nás zajímají. Výpisy mohou být prováděny periodicky ve zvláštním procesu a tím si vlastně vytvoříme velice pružný zobrazovač stavu aplikace, toho co nás zrovna zajímá. Proměnné vidíme v tom tvaru, ve kterém je chceme interpretovat. Výpisy na monitor z procesoru Kit se doporučuje tvořit v alfanumerickém režimu. Grafika většinou zabere moc času procesoru. K procesoru KitV40 se dá připojit pouze osmibitová VGA karta s ISA sběrnicí přes kartu EXPPC00. K procesoru Kit386EXR se dá připojit VGA karta přes sběrnici PC104. K procesoru Kit188ER nelze připojit žádnou VGA kartu – je nutno se spokojit s laděním přes systémový port sériové komunikace. I přes tuto komunikaci lze pomocí patřičných knihoven a připojeného PC zobrazovat výpisy, jako kdyby byly v aplikaci prováděny přímo na VGA monitor. Platí zde ale jistá omezení, která na druhou stranu mají i jisté výhody této „VGA přes COM“. Pro tento účel byl vytvořen balík knihoven CrtCom. Aplikaci v procesoru KitXxx nelze krokovat.

d. V aplikaci vytvoříme další proces KernelMonitor s nejvyšší prioritou a zavedeme ji do procesoru Kit. Aplikace se o něco zvětší a zpomalí oproti reálnému běhu. KernelMonitor umí komunikovat s programem RTD a umí pozastavovat procesy, zjišťovat jejich stav a zjišťovat obsah proměnných. Pokud je aplikace nahraná v paměti RAM procesoru, umí KernelMonitor i nastavit BreakPointy do programu. Vše je podrobně popsáno v dokumentu o RetosDebuggeru.

e. Pokud řídicí systém Kit má připojen nějaký terminál (např. Term10), můžeme pro výpis proměnných použít obrazovku tohoto terminálu a pracovat jako v bodě c.

9. Několik poznámek k tvorbě aplikačního programu

Následující podkapitoly objasňují několik důležitých informací o programování procesorů KitXxx pomocí knihoven ***SofCon***.

9.1. RETOS – knihovna Kernel

Tato kapitola předpokládá alespoň základní znalosti o.s.RETOS.

Operační systém reálného času RETOS poskytuje prostředky pro pseudoparalelní běh více úloh (procesů), které obsluhují danou aplikaci. (Například jeden proces tvoří automat řídící stroj, druhý proces komunikuje s uživatelem, třetí s nadřízeným počítačem a navzájem komunikují přes pomocné globální proměnné, nebo přes schránky RETOSu.) Uživatel ve své aplikaci o.s.RETOS může, ale také nemusí použít. Z knihovních modulů používají RETOS pouze terminálové a regulační knihovny. I při použití těchto knihoven se však dá RETOS vynechat. Pro aplikaci není nutné používat naráz všechny prostředky tohoto operačního systému reálného času. Například se běžně používají prostředky pro paralelismus, ale mnohem méně často

schránky operačního systému. Je totiž mnohem jednodušší komunikovat mezi úlohami přes obyčejné proměnné PASCALu, než přes schránky operačního systému.

Při použití RETOSu se trochu zamlží současnost dějů (procesů). Každý proces má totiž svou nastavitelnou prioritu (statickou a dynamickou). Lze použít dvě základní strategie přidělování priorit.

1. Úlohám lze nastavit různou statickou prioritu. Potom úloha s vyšší prioritou se musí vzdávat svého času ve prospěch úlohy s nižší prioritou například tím, že volá Wait(1). Pokud tak neučiní, úlohy s nižší prioritou nebudou obsluhovány. Ovšem úloze s vyšší prioritou to dává možnost pouštět čas pouze tehdy, když si to může dovolit a má jistotu, že každý tick RETOSu bude vyvolaná.
2. Úlohy mají stejnou statickou prioritu a liší se prioritou dynamickou. To zabezpečuje spravedlivé přidělování času, ovšem pokud nechce být úloha rušena, musí si na čas zvednout prioritu, nebo dočasně zakázat přepínání úloh. Tento mód sice zaručuje spravedlivé přidělování času, ale ne úplně pravidelné.

9.2. Přerušení

9.2.1. Přerušení od periferních desek

Od periferních desek jsou na sběrnici IOBUS zavedena hardwarová přerušení INT3 a INT4. Těmto signálům odpovídá vždy určité IRQ přerušení podle daného procesoru (viz následující tabulka).

	KitV40	**Kit386EXR**	**Kit188ER**
COM1 (COMA)	-	IRQ 4 – INT 0Ch	IRQ E – INT 0Eh
COM2 (COMB)	-	IRQ 3 – INT 0Bh	IRQ F – INT 0Fh
IOBUS INT3	IRQ 3 – INT 0Bh	IRQ 5 – INT 0Dh	IRQ C – INT 0Ch
IOBUS INT4	IRQ 4 – INT 0Ch	IRQ 2 – INT 71h	IRQ D – INT 0Dh

V tabulce je uvedeno vždy číslo IRQ a odpovídající číslo vektoru přerušení. Jelikož Kit386EXR obsahuje 2 standardní COM porty, musely být signály INT3 a INT4 ze sběrnice IOBUS přivedeny na jiný IRQ než u procesoru KitV40.

Tato přerušení používají zpravidla komunikační desky (např. IOCOM, IOPCOM, IOCAN), které obsluhují příslušné komunikační knihovny (zadává se v nich IRQ údaj).

9.2.2. Přerušení od systémového časovače – knihovna Tick

Knihovna **Tick** umí pracovat s přerušením od systémového časovače Int08, který standardně vyvolává přerušení jedenkrát přibližně za 55ms. Knihovna Tick umí toto přerušení modifikovat, zrychlovat či zpomalovat a zavěšovat na něj další úlohy.

9.2.3. Nemaskovatelné přerušení NMI

Nemaskovatelné přerušení je obslouženo BIOSem, ale prázdnou procedurou, která nic nedělá. Pokud aplikace vyžaduje obsluhu tohoto nemaskovatelného přerušení, které je vyvolána při poklesu napájecího napětí (při vypnutí řídicího systému), může použít knihovnu **uNmInt**.

9.3. Sériová komunikace

Pro sériovou komunikaci přes UART RS232, RS485, telefonní či GSM modem apod. byla vytvořena celá řada knihoven **ChnXxx**. Pro pochopení stavby těchto knihoven a jejich používání je nutné se seznámit s knihovnou **ChnVirt**, která definuje jednotné rozhranní a používání všech komunikačních knihoven.

V následující tabulce je přehled komunikačních knihoven nejnižší (fyzické) vrstvy pro jednotlivé porty řídicích systémů Kit.

KitV40		
COM (konektor X5 – RS232) (konektor X4 – TTL)	IO adresa i IRQ jsou již zahrnuty v příslušné knihovně	ChnV40, ChnV40_, ChnV40T
Kit386EXR		
COM1 (konektor X4 – RS232*)	ADD=$3F8 IRQ=4	ChnCom, ChnCom2, ChnComPB, ChnComBR, ChnComT
COM2 (konektor X6 – RS232*)	ADD=$2F8 IRQ=3	
Kit188ER		
COMS (konektor X2 – RS232)	IO adresa i IRQ jsou již zahrnuty v příslušné knihovně	Chn188
COMA (konektor X6 – RS232*)	ADD=$2100 IRQ=$E	ChnCom, ChnCom2, ChnComPB, ChnComBR, ChnComT
COMB (konektor X7 – TTL)	ADD=$2108 IRQ=$F	

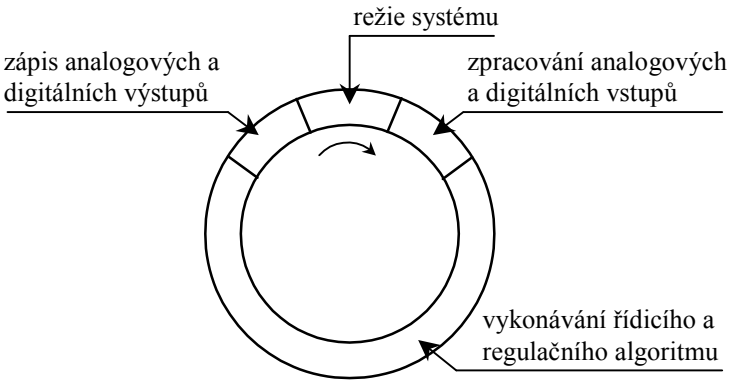
* jedná se o standardní zapojení, které se může propojkama podle příslušné HW dokumentace změnit na TTL

Ke všem procesorovým deskám Kit lze přes sběrnici IOBUS připojit jakoukoliv komunikační desku IOCOM, IOPCOM, IO485 apod. Všechny tyto desky se obsluhují pomocí komunikačních knihoven nejnižší (fyzické) vrstvy ChnCom, ChnCom2, ChnComPB, ChnComBR, ChnComT. Je nutno nastavit správně adresu (ADD) COM portu a číslo přerušení (IRQ) – viz kapitola „9.2.1 Přerušení od periferních desek“.

Pro sériovou komunikaci přes ETHERNET byly vytvořeny knihovny **CoXxx**. Pro pochopení stavby těchto knihoven a jejich používání je nutné se seznámit s knihovnou **CoBase**.

9.4. Automaty

Programátoři zvyklí programovat PLC automaty mohou způsobem ala PLC naprogramovat jednotku (unit) obsluhující hardwarové úlohy. Tuto jednotku, je dokonce výhodné programovat jako automat. Tato jednotka může mít svůj vlastní proces RETOSu. Struktura takovéhoto automatu může vypadat zhruba následovně:



V jazyce Pascal tomu může odpovídat následující algoritmus:

```
Begin
  Potřebné inicializace, v případě použití Retos inicializace jádra
  a procesů
  ...
  Repeat
    ReadDigInpust; {čtení digitálních vstupů}
    ReadAnlInputs; {čtení analogových vstupů}
    ...
    řídicí algoritmus
    ...
    ReadDigInpust; {zápis digitálních výstupů}
    ReadAnlInputs; {zápis analogových výstupů}
    Wait(1); {v případě použití RETOS pro uvolnění dalších procesů}
  Until ...;
  ...
End.
```

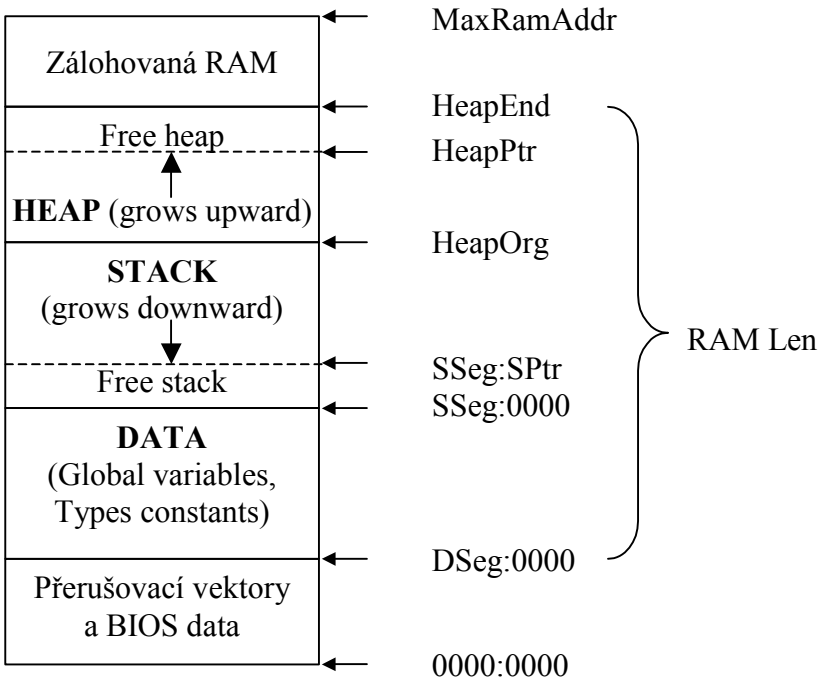
Při čtení digitálních nebo analogových vstupů bývá zpravidla potřeba provádět jejich filtraci pro omezení šumu. Potom je nutno tyto vstupy číst podstatně častěji a zejména pravidelněji, než umožňuje RETOS. Proto je vhodné použít pro čtení vstupů vlastní přerušovací rutiny pro systémový časovač (nebo ještě lépe využít možností knihovny Tick - viz kapitola „9.2.2 Přerušení od systémového časovače – knihovna Tick“) a v těchto rutinách provádět požadované filtry.

Pro zajištění konzistence načtených dat pro celý jeden úsek řídicího algoritmu je nutné načtené a vyfiltrované hodnoty uložit do pracovních proměnných, jejichž obsah zůstane během celého jednoho cyklu řídicího algoritmu konstantní. V případě používání načtených vstupů přímo by se totiž mohlo stát, že v průběhu vykonávání jednoho cyklu řídicího algoritmu dojde k vyvolání přerušovací rutiny od časovače, která provede nové načtení dig. či an. Vstupů a poté by se pokračovalo v řídicím algoritmu s jinými (novými) hodnotami vstupů. Ukázku zajištění konzistence ukazuje následující algoritmus:

```
Var DigInputs:...; {obraz uložených dig. vstupů}
...
Begin
   Potřebné inicializace, v případě použití Retos inicializace jádra
   a procesů
   ...
   {Inicializace driveru systémového časovače, jeho zrychlení na
   55/5, tj. 10ms a přiřazení čtení dig. vstupů s touto periodou
   v proceduře ReadDigInputs. Tato procedura bude načtené vstupy
   ukládat např. do proměnné LocDigInputs}
   InitTick;
   PushfCli; UserTick1:=@ReadDigInputs; Popf;
   SetTickDivider(11);
   ...
   Repeat
     PushfCli;
     DigInpust:=LocDigInputs; {zkopírování nově načtených digitálních
                               vstupů pro řídicí algoritmus}
     Popf;
     ReadAnlInputs; {čtení analogových vstupů}
     ...
     řídicí algoritmus pracující se vstupy DigInputs.
     ...
     ReadDigInpust; {zápis digitálních výstupů}
     ReadAnlInputs; {zápis analogových výstupů}
     Wait(1); {v případě použití RETOS pro uvolnění dalších procesů}
   Until ...;
   ...
End.
```

9.5. Paměť RAM

Paměť RAM procesorů KITXxx je zálohovaná baterií. Ta umožňuje uchování dat i při vypnutém napájení. Paměť RAM lze rozdělit na následující části:



DSeg – (Word) Segment adresy datového modulu. Nastavuje se v RTD resp. KitLoaderu na standardní hodnotu $00B0.

SSeg – (Word) Segment adresy začátku Stack (zásobníku).

SPtr – (Word) Offset adresy udávající začátek obsazeného Stack.

HeapOrq – (Pointer) Adresa začátku Heap (haldy) a zároveň konce Stack (zásobníku).

HeapPtr – (Pointer) Adresa konce zaplněného Heap a zároveň adresa začátku volného Heap.

HeapEnd – (Pointer) Adresa konce Heap. Vypočte se v RTD/KitLoader podle velikosti RAM modulu od které se odečte velikost STACK a DATA. Např. pokud je STACK = 16KB, DATA = 32KB a velikost RAM modulu se v RTD/Kitoader nastaví na $1E0000, bude velikost Heap = 122880-32768-16384 = 73728 = $12000 a HeapEnd bude mít tudíž hodnotu $12B0:0000.

RAM Len – Délka datového modulu. Nastavuje se v RTD resp. KitLoaderu dle potřeb dané aplikace.

MaxRamAddr – Maximální absolutní (lineární) adresa přímo adresovatelné RAM. Tj. pokud je KitXxx osazen 128KB RAM, nastaví si aplikace MaxRamAddr = $20000. Pro Kit188ER (který je osazen 1MB RAM) je ale tato hodnota dána hranicí mezi RAM a ROM v konfigurační tabulce BIOSu. Pokud je například tato hranice nastavena na hodnotu $7000:0000, je velikost přímo adresovatelné RAM 448KB a aplikace by měla nastavit MaxRamAddr = $70000.

DSeg, SSeg, SPtr, HeapOrg, HeapPtr a HeapEnd jsou globální proměnné jednotky Systém.tpu a jsou použitelné v řídicí aplikaci. Proměnnou MaxRamAddr si většinou nadefinuje každá aplikace sama a nastaví podle velikosti osazené RAM.

Zjištění velikosti jednotky DATA provede automaticky překladač podle počtu a velikostí globálních proměnných. Tuto velikost lze zjistit v RTD resp. KitLoaderu.

Velikost jednotky STACK se nastavuje standardními direktivami překladače Pascal (`{$M StackSize,HeapMin,HeapMax}`). Lze ji rovněž zjisti v RTD resp. KitLoaderu.

Velikost jednotky HEAP **nelze** nastavit standardními direktivami překladače Pascal jako v případě STACK. Velikost HEAP je dána jako RAM Len – velikost DATA – velikost STACK, tj. velikost datového modulu mínus velikost STACK mínus velikost DATA. Nelze ji jednoduše zjistit v RTD resp. KitLoaderu (tam se její velikost zobrazuje vždy jako $00000).

Jednotky DATA, STACK a HEAP obsluhují standardní funkce jazyka Pascal. Zálohovanou paměť RAM, která se používá pro globální proměnné a struktury, jejichž obsah se zachovává i po vypnutí napájení, Pascal standardně neobsluhuje. Tj. pokud chce programátor umístit do této části paměti své proměnné, musí je na příslušnou adresu naadresovat absolutně.

Např. `var MyParams : tMyParams absolute $1A00:$0000;`.

Zde je nutné dát pozor, aby tyto globální proměnné

a) Nezasahovaly mimo oblast přímo adresovatelné RAM, tj. mimo MaxRamAddr.

b) Nezasahovaly do části HEAP (samozřejmě ani do STACK a DATA).

c) Nepřekrývaly se navzájem.

Do zálohované paměti se většinou ukládají technologické parametry stroje, Restart struktura, archivy apod.

Následující příklad ukazuje nastavení pro KitV40 s 128KB RAM ze které je 30KB vyhrazeno pro zálohovanou paměť a 8KB pro Stack. V horní části zálohované paměti na nadefinována struktura Glb s globálními proměnnými

```
{$M 8192,xxx,xxx} {xxx znamená libovolné číslo, jelikož
                   velikost HEAP se takto nenastavuje}
type
  pGlb = ^tGlb;
  tGlb = record
    ... {zálohované položky}
  end;
const
  MaxRamAddr = $20000; {maximální adresa 128KB RAM}
  GlbAbsAddr = MaxRamAddr - SizeOf(tGlb); {absolutní adresa
                                           Glb}
  SegGlbAddr = GlbAbsAddr shr 4;
  OfsGlbAddr = GlbAbsAddr and $F;
  Glb : pGlb = Ptr(SegGlbAddr, OfsGlbAddr); {ukazatel na Glb}
```

Struktura Glb se poté již v aplikaci pro Kit nesmí alokovat příkazy New či GetMem.

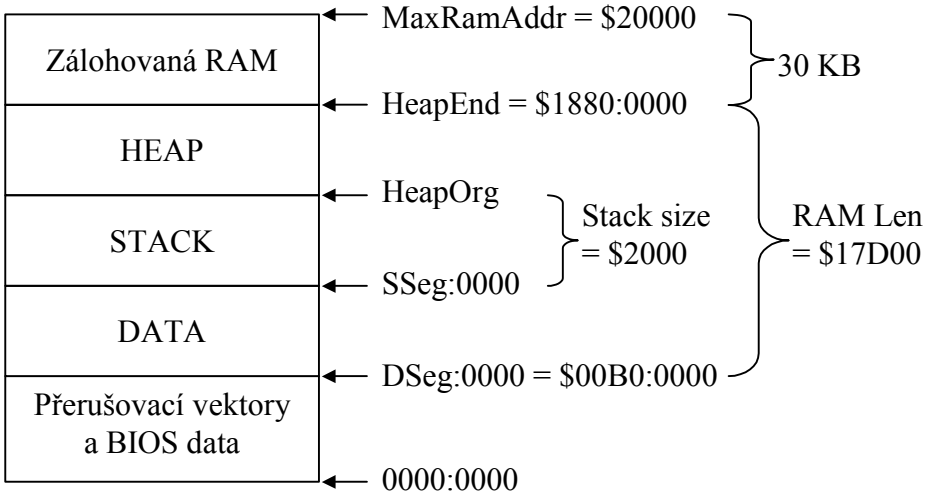
Při překladu aplikace pro PC je možno data v zálohované RAM simulovat ukládáním do souboru na disk. Ukazatel na Glb se poté nadefinuje jako `Glb : pGlb = nil;`, poté se zavolá `New(Glb);` - vytvoří se prostor na HEAPu PC, a nakonec se do Glb^ načtou data z uloženého souboru na disku.

V RTD resp. KitLoaderu se nastaví v „Memory design“ datový modul RAM na: Segment = $00B0, Len = $17D00 a vloží se do něj jednotky STACK, DATA a HEAP. Tím vznikne pro zálohovanou RAM prostor 30KB.

V programu dále můžeme provést kontroly překryvů pamětí:

```
if GlbAbsAddr + SizeOf(tGlb) > MaxRamAddr then Chyba;
if GlbAbsAddr < (longint(Seg(HeapEnd)) shl 4) + Ofs(HeapEnd)
    then Chyba;
```

Přehledné rozvržení paměti RAM pro tento příklad ukazuje následující obrázek:



Zálohování RAM baterií také způsobuje, že globální proměnné (v DATA segmentu) nejsou při spuštění programu vynulovány, jak to bývá v PC. Naopak po prvním spuštění programu už mají danou hodnotu. Například jednou vytvořený dynamický objekt je už vlastně vytvořený trvale. Jeho ukazatel i proměnné, pokud je destructor nezruší, zůstávají v paměti zachovány i po vypnutí napájení. Je zapotřebí mnohem více dbát na inicializaci všech proměnných než v aplikacích pro PC. Jako dobrá metoda se jeví místo obyčejných proměnných používat typové konstanty (`const Variable : Type = Value;`).

Naopak proměnné, které zaručeně chceme zachovat i po výpadku napájení a při různých haváriích programu je výhodné umístit na absolutní adresu úplně mimo oblast RAM modulu programu, tj. do zálohované RAM.

9.6. Paměť Flash

Do paměti FLASH lze zapisovat i v průběhu běhu aplikace a to pomocí speciálních funkcí systémové knihovny **AtmFlash**. Tím lze do této paměti ukládat i některá data aplikace – např. vyžadujeme-li větší bezpečnost jejich uložení než poskytuje zálohovaná RAM. Jen je třeba dbát na maximální počet přepsání jednoho sektoru paměti a na pokus o přímý zápis do paměti Flash jako do RAM. Ochrana tohoto přímého zápisu totiž způsobí načtení nesmyslných dat kódu aplikace a tím i její pád - reset.

9.6.1. Nahrání nové verze aplikace do paměti Flash

Jsou možné dva způsoby: hardwarový a softwarový.

Hardwarový se provádí pomocí příslušných zkratovacích propojek na procesorové desce KitXxx. Nejprve rozpojíme propojku obvykle s názvem App/Bios, a poté provedeme krátkým spojením propojky Rst reset procesoru. Tím se spustí aplikace Bios monitor, pomocí něhož můžeme po komunikaci z RTD resp. KitLoaderu nahrávat novou verzi řídicí aplikace. Nesmíme zapomenout opět spojit

propojku App/Bios, jinak by při příštím resetu či vypnutí a zapnutí napájení nenaběhla řídicí aplikace, ale Bios monitor.

Softwarový způsob se používá v případě, že nemáme přístup k propojkám pro hardwarový způsob (např. díky ztíženým montážním podmínkám, zavřenému rozvaděči apod.). Přechod do Bios monitoru musí zajistit (např. na příkaz z klávesnice terminálu) sama řídicí aplikace. Nejprve pomocí funkce **AtmDestruct** z knihovny **AtmFlash** poruší první spouštěcí ROM modul aplikace (ten který obsahuje jednotku --Loader--). Potom provede reset procesoru (např. příkazem `asm cli end; repeat until false;`). Tím se spustí BIOS, který při prohledávání Flash paměti nenajde spustitelnou aplikaci (jelikož jsme ji příkazem AtmDestruct poškodili) a proto spustí Bios monitor. Pomocí něho po komunikaci z RTD resp. KitLoaderu nahrajeme novou verzi řídicí aplikace.

9.7. WatchDog

Obvod WatchDog umístěný na procesorové desce je občerstvovaný z Biosu v proceduře obsluhy systémového časovače Int 08h. To zajistí jeho spolehlivé obsloužení i v aplikačním programu. Ovšem při zhroucení aplikačního programu se málokdy poškodí také obsluhy přerušení a WatchDog zůstane dále občerstvován. Lepší je, pomocí knihovního modulu si vytvořit vlastní obsluhu obvodu WatchDog, ovšem až v odladěné aplikaci.

9.8. RunTime Errors (Chyby za běhu programu)

Borland Pascal může generovat runtime errors neboli chyby za běhu programu. Jedná se o chyby způsobené zpravidla nedbalostí programátora (např. přetečení rozsahu, aritmetické přetečení, odkaz na neexistující objekt, přetečení zásobníku apod). Je výhodné v aplikaci nechat kontroly těchto chyb zapnuté (viz příslušné direktivy překladu $R $Q $S $O), poněvadž to nijak významně program nezpomaluje ani nezvětšuje a přitom je program hlídán.

Zjistí-li některá z kontrol chybu programu, vygeneruje příslušný RunTime Error, což se projeví ukončením programu a vyvoláním závěrečné procedury ExitProc. V těle této ExitProc může programátor danou chybu zachytit v proměnné ExitCode (číslo chyby) a ErrorAddr (adresa chyby).

Pozn: ExitProc se vyvolá i při standardním regulérním ukončení programu (např. příkazem Halt). V takovém případě je ExitCode = 0 (žádná chyba) a ErrorAddr = nil.

Ve verzi pro KitXxx je třeba v Exit proceduře zapisovat chybovou adresu, číslo chyby a také i datum a čas vzniku do zálohované oblasti RWM a mít nástroj, kterým se dá tato oblast číst. Je nutno mít na paměti, že při ukončení, tj. spadnutí programu na RunTime Error spustí BIOS daný program znovu. Ten může opět skončit chybou. Dochází tak k opakovanému padání daného programu. Pokud program neustále padá, můžeme ho příslušnou propojkou přepnout do BIOS Monitoru a uloženou chybu z RAM přečíst pomocí RTD. Z takto přečtené adresy chyby opět pomocí RTD zjistíme ve které jednotce (unit) a na které řádce k chybě došlo.

Ve startovacích aplikacích je jak tato zálohovaná struktura, tak exit procedura založená.

9.9. Ladicí výpisy na VGA

Ke **KitV40** lze připojit 8mi bitovou VGA kartu do ISA slotu přes přídavný modul EXPPC00. Poté je v paměťovém prostoru vyhrazen prostor od adresy 0A0000h do 0C7FFFh, do kterého lze přistupovat standardním způsobem (v textovém módu pomocí jednotky CRT a příkazů Write apod., v grafickém módu pomocí jednotky Graph). Tím je možno aplikaci obohatit o pomocné ladicí výpisy.

Ladění aplikace pomocí těchto výpisů je rychlé, ale použitelné pouze ve fázi vývoje na pracovišti. V reálných podmínkách většinou není možnost připojení přídavného modulu EXPPC00, navíc se toto připojení musí provádět při vypnutém napájení.

Pokud programátor nepotřebuje výpisy na VGA a tudíž ani vyhrazený paměťový prostor 0A0000h - 0C7FFFh, lze tento prostor rovněž využít pro kód vlastní řídicí aplikace. K tomu je ale nutný jiný obvod GAL v patici U5 a aplikace **NESMÍ** žádným způsobem použít jakýchkoliv služeb VGA (tj. ani žádný příkaz Write).

Ke **Kit386EXR** lze připojit VGA kartu přes sběrnici PC104. Poté je v paměťovém prostoru vyhrazen prostor od adresy 0A0000h do 0C7FFFh, do kterého lze přistupovat standardním způsobem (v textovém módu pomocí jednotky CRT a příkazů Write apod., v grafickém módu pomocí jednotky Graph) stejně jako na KitV40.

Pokud programátor nepotřebuje výpisy na VGA, lze tento prostor rovněž využít pro kód vlastní řídicí aplikace. K tomu je nutné pouze jiné konfigurační nastavení tabulky BIOSu, které se provede v RTD/KitLoader nastavením položky Cut Size na 0KB a položky Low Size na 256KB. Aplikace poté **NESMÍ** žádným způsobem použít jakýchkoliv služeb VGA (tj. ani žádný příkaz Write).

Ke **Kit188ER** nelze připojit žádnou standardní VGA karta. Přesto programátor aplikace není o ladicí výpisy na VGA ochuzen. Musí však aplikaci doplnit o balík knihoven **CrtCom** (viz příslušný manuál k těmto knihovnám), které zajišťují výpisy na obrazovku vzdáleného PC s aplikací CrtTerminal přes sériovou komunikaci. Tento způsob má jistá omezení (nelze použít grafiku, přenos dat je pomalejší) ale i nesporné výhody (menší nároky na paměť kódu, možnost připojení k CrtTerminalu za chodu řídicího systému bez nutnosti v praxi složitého připojení VGA karty).

Použití knihoven CrtCom je možné i na řídicích systémech KitV40 a Kit386EXR bez vyhrazeného prostoru pro VGA, tj. větší prostor pro vlastní kód aplikace. Tím (jak již bylo řečeno) bude výpis na vzdálený CrtTerminál o něco pomalejší, ale zato bude mít podstatně menší nároky na kód a k řídicímu systému se bude moci připojit sériovou komunikační linkou i za běhu aplikace. Vyžaduje to ovšem pro tyto účely jeden volný COM port na řídicím systému Kit.

10. Startovací aplikace

Na CD disku firmy *SofCon s.r.o.* se nacházejí příklady k jednotlivým knihovnám (v adresářích Examples u knihoven LIB, LIBT a LIBV). Jedná se například o příklady v adresářích BEG_T10, BEGT10V, BEGN_T01 a další. V těchto adresářích se nacházejí založené minimální aplikace s terminály Term 10 a Term 01. Aplikace obsahují úvodní bitmapu, první menu obrazovky, přechod do servisní obrazovky chráněné heslem, obrazovku nastavení hesla a obrazovku výpisu RuntimeErrors. Aplikace mají založené hlavní unity a procesy. Mají založenou strukturu Glb (zálohovaná struktura s technologickými parametry) a její inicializaci. V Exit proceduře je zápis adresy a čísla RunError do Glb struktury. V servisní obrazovce je možné běžící aplikaci softwarově zrušit, přejít do Bios Monitoru a nahrát pomocí RTD či KitLoader novou verzi. V DOS verzi se používá simulátor terminálu vypisující na monitor. Aplikace mají založený soubor Sets.Inc s podmínkami překladu.

Doporučujeme při zakládání nové aplikace vycházet z těchto minimálních aplikací a rozvíjet je.