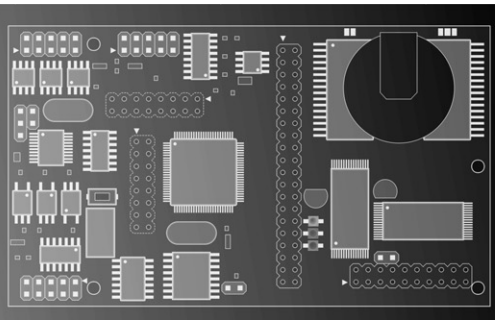
ELSACO, Jaselská 177
28000 KOLÍN, CZ
tel/fax +420-321-727753
http://www.elsaco.cz
mail: elsaco@elsaco.cz



Stavebnice PROMOS Line 2

# MCPU-01

## Mikroprocesorový modul – popis, programování, BIOS

*Technický manuál*

*19. 07. 2001*



Účelová publikace ELSACO

**ELSACO, Jaselská 177, 280 02  Kolín 3**
Tel./fax/modem: 321 727 753 / 321 727 759
Internet: **www.elsaco.cz**

**Připomínky:** vondruska@elsaco.cz

# 1 MCPU-01 – MIKROPROCEROVÝ MODUL PROMOS LINE 2

## 1.1 Obsah tohoto manuálu

Manuál popisuje možnosti procesorového modulu MCPU-01, jeho použití, zapojení a využití programových služeb firmware. Součástí manuálu není vlastní popis procesoru, jeho vnitřní struktury, zapojení a instrukčního souboru ani uživatelský popis vývojového prostředí TOPAS.Popis procesoru je dostupný na adrese http://www.toshiba.com/taec/components/Datasheet/TMP95CS64F-C265F.shtml.

## 1.2 Úvod

MCPU-01 je univerzální procesorový modul, jehož blokové schéma je na obr. 1. Byl vyvinut jako základ centrálních jednotek CCPU stavebnice PROMOS line 2, stejně dobře však může být použit samostatně jako procesorový modul zastavěný do hostitelské desky nebo jako samostatný komunikační procesor. Výkonný šestnáctibitový procesor Toshiba TMP95C265F, velký objem zálohované statické paměti RAM, FLASH EPROM a dobré komunikační možnosti předurčují modul pro výstavbu řídicích a regulačních jednotek, jednoúčelových přístrojů a komunikačních koncentrátorů. Modul tvoří ucelený mikropočítačový blok se všemi nezbytnými obvody včetně lithiové zálohovací baterie.

## 1.3 Základní charakteristiky modulu

**Procesor TMP95C265F**

- taktovací kmitočet 14,746 nebo 24,576 MHz,
- 2 KB interní RAM,
- 2 programovatelné výstupy CS pro vnější periferie,
- lineární adresování 24 bitů (16 MB adresový prostor),
- 4 kanály DMA (bloky 64 KB, paměť–paměť, paměť–IO),
- 8× 8bit (4× 16bit) timer s možností přerušení,
- 2× 16bit timer s možností přerušení,
- 3× sériový kanál,
- AD 8 kanálů 0 ÷ 5V s rozlišením 10 bitů,
- 2× DA 0÷5V s rozlišením 8 bitů.

**Paměť**

- statická zálohovaná RAM 256 KB nebo 1 MB,
- FLASH EPROM 256 KB nebo 1 MB.

**RTC**

- obvod RTC4553 zálohovaný lithiovou baterií,
- přesnost 5 ppm.

**WatchDog**

- interní WD procesoru s programovatelnou časovou konstantou (generuje INT),
- externí WD 1,6 s (generuje RESET).

**Power fail**

- vstup s komparátorem pro kontrolu napětí před stabilizátorem (NMI),
- komparátor pro kontrolu stavu baterie.

**Sériové linky**

- jsou využity 3 interní kanály procesoru,
- kanál 0 – asynchronní RS-232 + volitelně multistandard infrared (podporuje standardy IrDA, HPSIR, ASK&TV),
- kanál 1 – asynchronní RS-422/485 s galvanickým oddělením, vnější napájení 5 V galvanicky oddělené strany,
- kanál 2 – asynchronní RS-422/485 + CSI/O synchronní RS-422 (Din, Dout, obousměrný Clk, STB, RST).

**Doplňkové možnosti – zákaznické modifikace**

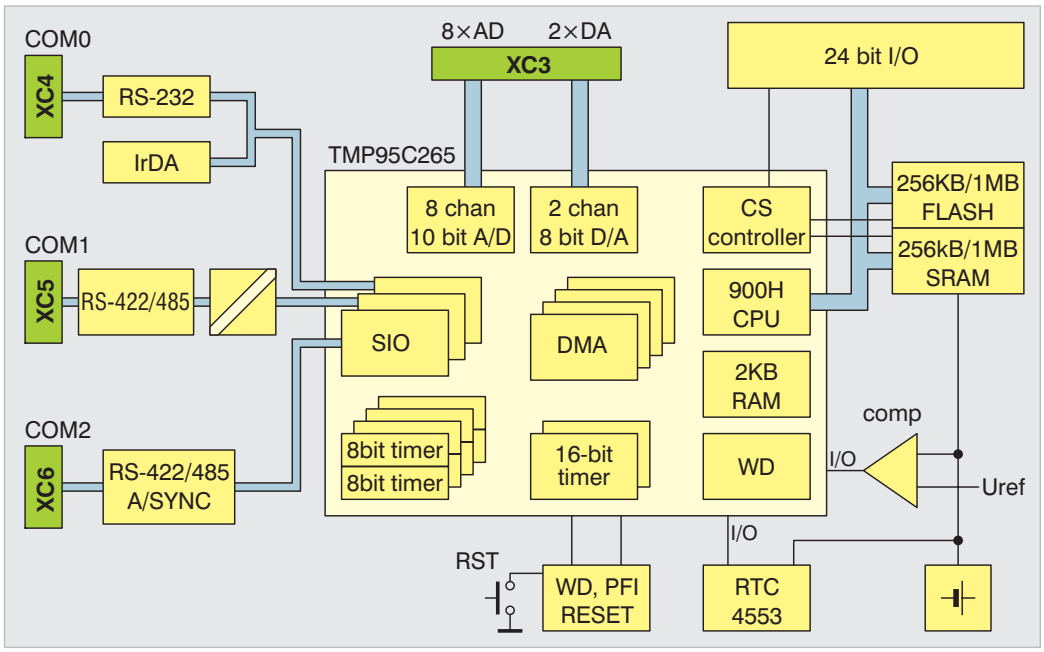
- EEPROM 1× 512 Byte + 1× 512 B ÷ 32 KB,
- elektronické výrobní číslo,
- teplotní snímač,
- měnič pro napájení sériové linky s GO,
- IR rozhraní pro komunikační kanál 0.

**Ostatní**

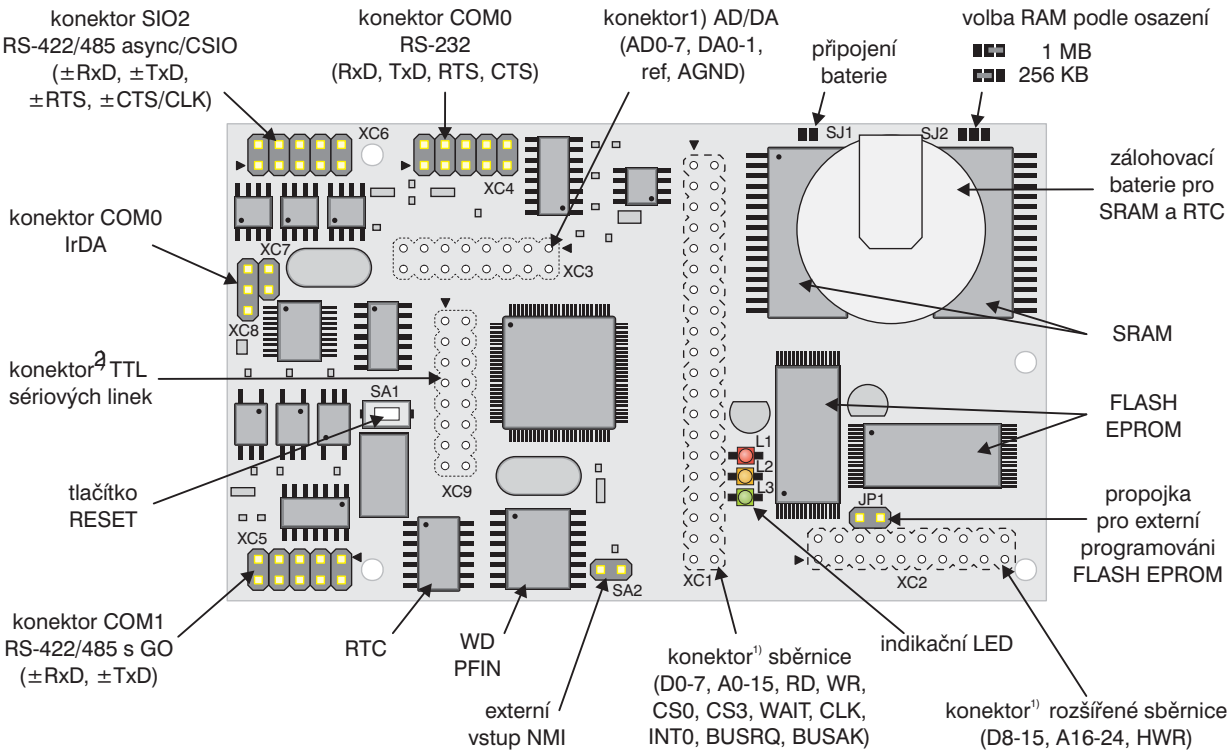
- napájecí napětí 5 V ± 5 %,
- proudový odběr max. 100 mA,
- proudový odběr s měničem pro galvanicky oddělenou sériovou linku max. 200 mA,
- rozsah pracovních teplot –10 ÷ 50 °C,
- rozměry desky 100,3 × 58,4 mm.

## 1.4 Vývoj programového vybavení

Pro vývoj programového vybavení se používá firemní balík programů Toshiba TOPAS 900, který obsahuje ANSI C compiler, assembler, linker a on line C source debugger. Prostředí běží na standardním PC a procesorový modul je připojen sériovou




*Obr. 1: Blokové schéma modulu MCPU-01*




*Obr. 2: Horní strana modulu (strana součástek)*

linkou. Prostředí umožňuje sestavení programu, zavedení programu do paměti mikropočítače a ladění na úrovni C nebo ASM.

Vestavěný firmware modulu zajišťuje upload programu ze sériové linky a uložení do FLASH, spuštění programu, nebo spuštění monitoru pro C source debugger. Základní verze fimware také poskytuje služby pro obsluhu všech hardwarových komponentů modulu včetně sériových komunikací a obsahuje jádro reálného času TORTOS16.

## 1.5 Konektory a propojky

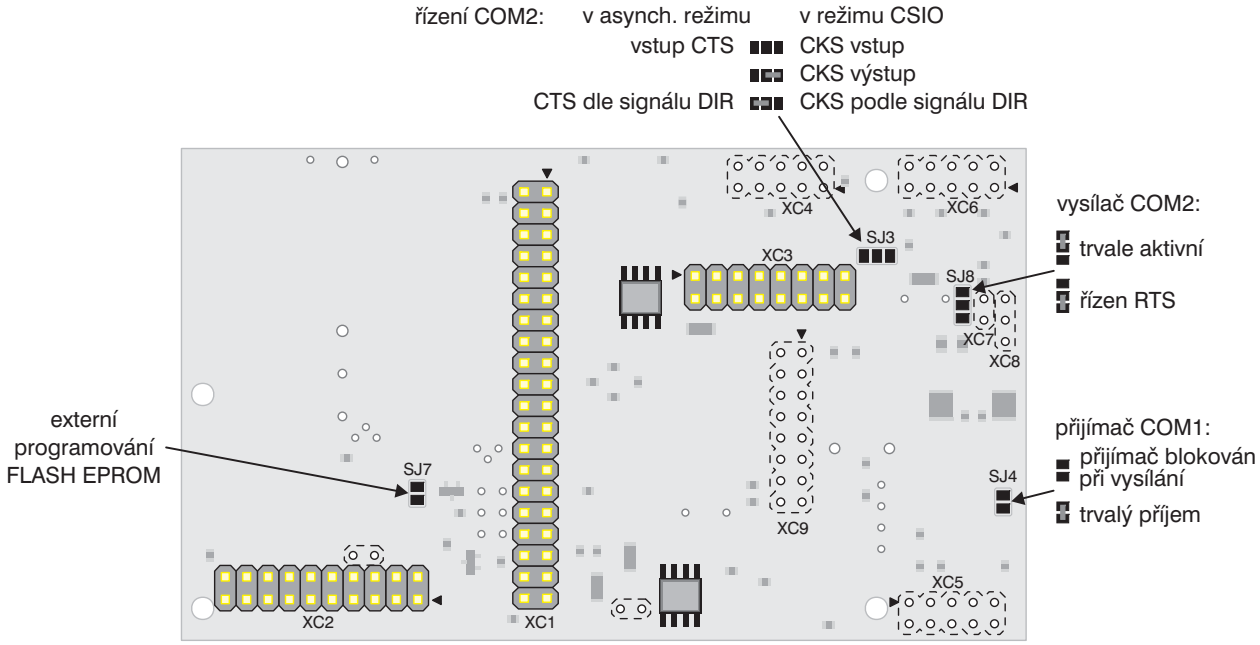
Rozmístění součástek na modulu uvádí obrázky 2 a 3. Spojení s hostitelskou deskou zajišťují konektory XC1 (základní část sběrnice procesoru) a XC2 (rozšířená sběrnice). Konektor XC3 umožňuje připojit analogové a binární vstupy a výstupy procesoru.

Konektor XC9 (je osazen pouze v některých verzích) umožňuje na hostitelskou desku připojit sériové linky, pokud nejsou využity připojovací konektory sériových rozhraní na horní straně desky.

Propojka SJ1 umožňuje odpojit baterii pro vymazání RAM nebo kontrolu zálohovacího proudu baterie. Propojka SJ2 je zapojena podle osazených obvodů SRAM. Pájecí propojky SJ3, SJ4 a SJ8 pro konfiguraci sériových rozhraní jsou také na spodní straně modulu.

Propojky JP1 a SJ7 umožňují přepojení výběrových signálů FLASH paměti a dovolují tak naprogramování FLASH z vnějšího zařízení. To je nezbytné při prvotním naprogramování firmware nebo po přepsání systémových programů paměti.

Na následující stránce je v tabulkách 1, 2 a 3 uvedeno zapojení jednotlivých konektorů.




*Obr. 3: Dolní strana modulu (strana konektorů)*

*Tab. 1: Možnosti připojení vstupů/výstupů*

| Použití | Konektor | Čísla pinů | Název | Poznámky |
|---|---|---|---|---|
| Analogové/binární vstupy | XC3 | 1…8 | PA0…7 | Pouze jako vstupy |
| Analogové/binární výstupy | XC3 | 13, 14 | DA0…1 | Pouze jako výstupy |
| Binární vstupy/výstupy | XC2 | 5…8 | A20…23 | Bez rozšiřující paměti na externí desce |
| Binární vstupy/výstupy | XC2 | 3, 4 | A18, A19 | Velikost paměti omezena na 256kB SRAM a 256kB Flash |
| Binární vstupy/výstupy | XC1 | 38 | BusRq | Bez použití externího DMA |
| Binární vstupy/výstupy | XC1 | 40 | BusAk | Bez použití externího DMA |
| Binární vstupy/výstupy | XC1 | 37 | /CS3 | Bez rozšiřující paměti na externí desce |
| Binární vstupy/výstupy | XC1 | 35 | /CS0 | Bez externích periferií |
| Binární vstupy/výstupy | XC1 | 34 | /Wait | Omezeno připojení pomalých periferií |
| Binární vstupy/výstupy | XC9 | 2 | L/K | Bez použití LED L1 a elektronického výrobního čísla |
| Binární vstupy/výstupy | XC9 | 15 | D/C | Moduly bez IrDA |
| Binární vstupy/výstupy | XC9 | 16 | DIR | Bez řízení směru přenosu SCLK na CSIO |
| Binární vstupy/výstupy | XC9 | 3 | CTS2 | Bez CTS na COM2 |
| Binární vstupy/výstupy | XC9 | 7, 8 | TxD1, RxD1 | Bez COM1 |
| Binární vstupy/výstupy | XC9 | 9 | RTS1 | Bez RTS na COM1 |
| Binární vstupy/výstupy | XC9 | 10, 11 | TxD2, RxD2 | Bez COM2 |
| Binární vstupy/výstupy | XC9 | 14 | RTS2 | Bez RTS na COM2 |

*Tab. 2: Zapoj. konektorů pro spojení s hostitelskou deskou*

| XC1 sběrnice | | | | XC2 rozšíření sběrnice | | XC9[2)] sériové linky TTL | | XC3 analog I/O | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Vbat | 2 | PFI | 1 | A16 | 1 | $\overline{\text{NMI}}$ | 1 | PA0 |
| 3 | +5V | 4 | GND | 2 | A17 | 2 | L/K | 2 | PA1 |
| 5 | A0 | 6 | A1 | 3 | A18 | 3 | CTS2 | 3 | PA2 |
| 7 | A2 | 8 | A3 | 4 | A19 | 4 | TxD0 | 4 | PA3 |
| 9 | A4 | 10 | A5 | 5 | A20 | 5 | RxD0 | 5 | PA4 |
| 11 | A6 | 12 | A7 | 6 | A21 | 6 | $\overline{\text{CTS0}}$ | 6 | PA5 |
| 13 | A8 | 14 | A9 | 7 | A22 | 7 | TxD1 | 7 | PA6 |
| 15 | A10 | 16 | A11 | 8 | A23 | 8 | RxD1 | 8 | PA7 |
| 17 | A12 | 18 | A13 | 9 | $\overline{\text{HWR}}$ | 9 | $\overline{\text{RTS1}}$ | 9 | VrefH |
| 19 | A14 | 20 | A15 | 10 | GND | 10 | TxD2 | 10 | VrefL |
| 21 | D6 | 22 | D7 | 11 | D14 | 11 | RxD2 | 11 | AVcc |
| 23 | D4 | 24 | D5 | 12 | D15 | 12 | GND | 12 | AVss |
| 25 | D2 | 26 | D3 | 13 | D12 | 13 | $\overline{\text{RTS0}}$ | 13 | DA0 |
| 27 | D0 | 28 | D1 | 14 | D13 | 14 | RTS2 | 14 | DA1 |
| 29 | +5V | 30 | GND | 15 | D10 | 15 | D/C | 15 | +5V |
| 31 | $\overline{\text{RD}}$ | 32 | INT0 | 16 | D11 | 16 | DIR | 16 | GND |
| 33 | $\overline{\text{WR}}$ | 34 | $\overline{\text{Wait}}$ | 17 | D8 | | | | |
| 35 | $\overline{\text{CS0}}$ | 36 | $\overline{\text{RST}}$ | 18 | D9 | | | | |
| 37 | $\overline{\text{CS3}}$ | 38 | $\overline{\text{BusRq}}$ | 19 | +5V | | | | |
| 39 | CLK | 40 | $\overline{\text{BusAk}}$ | 20 | GND | | | | |

*Tab. 3: Zapojení konektorů sériových linek*

| XC4 – COM0 RS-232 | | XC5 – COM1 RS-422 GO | | XC6 – COM2 RS-422/CSIO | | |
|---|---|---|---|---|---|---|
| 1 | | 1 | +RxD | 1 | +RTS | +WRS |
| 2 | | 2 | –RxD | 2 | –RTS | –WRS |
| 3 | RxD | 3 | | 3 | +TxD | +TxS |
| 4 | RTS | 4 | | 4 | –TxD | –TxS |
| 5 | TxD | 5 | | 5 | +RxD | +RxS |
| 6 | CTS | 6 | | 6 | –RxD | –RxS |
| 7 | | 7 | +TxD | 7 | Res | Res |
| 8 | | 8 | –TxD | 8 | GND | GND |
| 9 | GND | 9 | SG | 9 | +CTS | +CKS |
| 10 | +5V | 10 | +5V GO | 10 | –CTS | –CKS |

# 2 POPIS BIOSU A PROGRAMOVÁNÍ MCPU-01

**UPOZORNĚNÍ: Pro správné pochopení všech důležitých informací je nutné manuál přečíst celý!**

Veškeré dotazy, připomínky a nejasnosti konzultovat nejlépe telefonicky nebo pomocí elektronické pošty.

## 2.1 Rozdělení adresového prostoru

### 2.1.1 Registry interních periférií

`000000..00009f` Umístění dáno hardwarem, nelze změnit. Přímý přístup (čtení či zápis) na tyto adresy může způsobit nefunkčnost služeb BIOSu.

### 2.1.2 Interní RAM

`0000a0..00089f` Umístění dáno hardwarem, nelze změnit. Zcela využita BIOSem.

### 2.1.3 Externí periférie

`008000..00ffff` Rezervováno pro externí periférie. Vybíráno signálem CS0.

### 2.1.4 Onboard RAM

`700000..7fffff` Paměť RAM osazená přímo na desce o velikosti 256 kB nebo 1 MB. V případě menší varianty dochází k zrcadlení. Aktivace signálem CS1.

### 2.1.5 Externí paměť

`800000..efffff` Rezervováno pro rozšíření o max. 7 MB RAM nebo EPROM. Aktivace signálem CS3.

### 2.1.6 Onboard Flash EPROM

`f00000..ffffff` Paměť Flash EPROM osazená přímo na desce o velikosti 256 kB nebo 512 kB. V případě menší varianty dochází k zrcadlení. Posledních 64 kB je obsazeno BIOSem. Aktivace signálem CS2.

## 2.2 POST

Po RESETu vždy proběhne jednoduchý test pro zjištění velikosti a typu pamětí. Určí se rychlost procesoru porovnáním s hodinami reálného času. Zinicializuje se oblast systémových proměnných. Nastaví se komunikační parametry pro sériový kanál 0. Po ukončení POSTu se rozsvítí žlutá LED indikující vyčkávání na povely ze sériové linky. V případě neúspěchu některého dílčího testu svítí také červená LED. Po uplynutí definované doby (viz. Defaultní nastavení) se BIOS pokusí spustit uživatelský program.

Při nalezení uživatelského programu se rozsvítí zelená LED. Spuštění monitoru pro UDE debugger indikuje rychle blikající žlutá. Po navázání komunikace se její stav mění v závislosti na činnosti debuggeru. Červená a zelená LED je k dispozici uživatelskému programu.

## 2.3 Spouštění programů

Po POSTu čeká BIOS cca 5 sekund (volitelné, viz. Defaultní nastavení) na nějaký příkaz z kanálu 0 (RS-232) na rychlosti 38400Bd, 8 datových a jeden stop bit bez parity. Pokud v daném čase žádný platný příkaz nepřijde, pokusí se spustit uživatelský program. Uživatelský program musí začínat hlavičkou na adrese `0xf00000`:

```
org 0xf00000
  db "RUNADDR:"     ;signatura
  dl start          ;vstupní bod programu
  dl chksum         ;=start+'ANUR'+':RDD'
```

Při spuštění uživatelského programu je povoleno přerušení a ukazatel zásobníku je nastaven na nevelký interní zásobník systému. Je tedy nutné, aby uživatelský program v první řadě nastavil vlastní hodnotu do registru XSP.

Pokud BIOS na adrese `0xf00000` nenalezne hlavičku uživatelského programu, zkusí spustit firemní UDE debugger od Toshiby. Ten začíná hlavičkou na adrese `0xfe0000`:

```
org 0xfe0000
  db "UDE:"     ;signatura
  dl start      ;vstupní bod debuggeru
  dl chksum     ;=start+':EDU'
```

Pokud není nalezena ani tato hlavička, BIOS pokračuje v čekání na příjem příkazů.

## 2.4 BIOS

BIOS zajišťuje mapování paměti, nahrávání programů do Flash EPROM, poskytuje služby pro ovládání periférií (SIO, RTC, WD, IrDa apod.). Volitelně nabízí i RT jádro pro správu procesů, protokol ProfiBUS a případně i další služby. Obslužné rutiny jsou navrženy tak, aby je bylo možné volat jako C-funkce s parametry předávanými v registrech (podrobnosti viz. Dodatky – Volací konvence), výsledky vrací v registru XHL. Vstupní body jsou v rozskokové tabulce začínající od adresy `0xfe0000`. Je přiložen definiční soubor `tabs1.inc` jako include pro zdrojový text v asembleru a hlavičkový soubor `sluzby.h` pro jazyk C.

## 2.5 Implementované funkce

### 2.5.1 Přehled

Garantovány jsou pouze funkce popsané v tomto manuálu. Všechny případné další deklarace v hlavičkových souborech nebo programech jsou pouze testovací a jejich zachování v dalších verzích není zaručeno. Není-li uvedeno jinak, funkce vrací následující chybové kódy:

- **0** O.K., funkce proběhla v pořádku,
- **1** zařízení je zaneprázdněno,
- **2** požadovaný prostředek zamknut jiným procesem,
- **3** neplatná adresa zařízení/prostředku,
- **4** nepřípustná hodnota parametru(-ů).

### 2.5.2 Manipulace s pamětí

`void ldir(void *odkud, int kolik, void *kam)`

Zkopíruje blok paměti (rychlejší než `memcpy` ze standardní knihovny). Parametr `kolik` udává délku bloku v bajtech (0=65536).

`byte chksum(void *odkud, int kolik)`

Vrátí součet modulo 256 bloku bajtů. Parametr `kolik` udává délku bloku v bajtech (0=65536).

`long ramtop(void)`

Vrátí adresu posledního bajtu souvislé oblasti RAM nalezené při POSTu.

`long ramusr(void)`

Vrátí adresu prvního bajtu souvislé oblasti RAM použitelné uživatelským programem. Zásah do paměti pod touto adresou může způsobit nedefinované chování systému.

### 2.5.3 Ovládání LED

```
int red_LED(int stav)
int yellow_LED(int stav)
int green_LED(int stav)
```

Pro `stav=0` zhasne (off), jinak rozsvítí (on) LED odpovídající barvy. Vrací předchozí stav. Vzhledem k vícenásobnému využití pinů, na kterých jsou připojeny LED, může činností některých služeb (obsluha RTC, ext. WDT, sériové

EEPROM, HW sériového čísla, teploměru) dojít k narušení jejich stavu.

### 2.5.4 Obsluha přerušení

Následující služby dovolují přesměrovat obsluhu přerušení. Obslužné funkce musí zachovat obsahy všech registrů a končit instrukcí RETI. V C je toho možné dosáhnout následující deklarací:

```
void __interrupt obsluha(void);
```

V tomto případě překladač sám zajistí uložení všech potřebných registrů na zásobník při zahájení a jejich obnovení před vykonáním instrukce RETI. (*Pozn.:* Tento požadavek neplatí pro obslužné funkce nastavené pomocí funkcí **ser_oef**, **ser_ief**, **cnt_init** apod.) Funkce pro přesměrování vrací předchozí adresu nebo hodnotu -1 (**0xffffffff**), pokud došlo k chybě (pokusu o přesměrování nedovoleného vektoru).

```
long set_swi(int n,void __interrupt(*obsluha)(void))
```

Přesměruje obsluhu softwareového přerušení číslo **n** (1 až 7). RESET (SWI0) přesměrovat nelze.

```
long set_nmi(void __interrupt(*obsluha)(void))
```

Signál NMI je obvykle aktivován pouze při poklesu napájecího napětí. Uživatelská obsluha je volána teprve po provedení obsluhy systému.

```
long set_wdt(void __interrupt(*obsluha)(void))
```

Obsluha interního WatchDogu procesoru. BIOSem není využíván a po RESETu je zastaven.

```
long set_int(int n,int p, void(*obsl)(void))
```

Nastaví adresu obslužné rutiny přerušení **n** a přiřadí mu prioritu **p** (pokud má smysl).

***Softwarová:***

1 ... SWI0 (RESET)
2 ... SWI1
3 ... SWI2 (Invalid Opcode)
4 ... SWI3
5 ... SWI4
6 ... SWI5
7 ... SWI6
8 ... SWI7

***Hardwarová nemaskovatelná:***

9 ... NMI
10 ... INTWD (interní watchdog)

***Hardwarová maskovatelná s programovatelnou prioritou:***

- vstupní piny
  - 11 ... INT0
  - 12 ... INT1
  - 13 ... INT2
  - 14 ... INT3
  - 15 ... INT4
  - 16 ... INT5
  - 17 ... INT6
  - 18 ... INT7
  - 19 ... INT8
- čítače/časovače 8bit
  - 20 ... INTT0
  - 21 ... INTT1
  - 22 ... INTT2
  - 23 ... INTT3
  - 24 ... INTT4
  - 25 ... INTT5
  - 26 ... INTT6
  - 27 ... INTT7
- čítače/časovače 16bit
  - 28 ... INTTR8
  - 29 ... INTTR9
  - 30 ... INTTRA
  - 31 ... INTTRB
  - 32 ... INTTO8
  - 33 ... INTTO9
- sériové kanály
  - 34 ... INTRX0
  - 35 ... INTTX0
  - 36 ... INTRX1
  - 37 ... INTTX1
  - 38 ... INTRX2
  - 39 ... INTTX2
- A/D převodník
  - 40 ... INTAD
- MicroDMA
  - 41 ... INTTC0
  - 42 ... INTTC1
  - 43 ... INTTC2
  - 44 ... INTTC3

Přiřazení pořadových čísel odpovídá implicitním prioritám jednotlivých přerušení.

```
void set_ilev(int p)
```

Nastavení aktuální priority přerušení (instrukce **ei p**). Používat velmi obezřetně!

### 2.5.5 Ovládání sériových portů

Před voláním funkcí pro vysílání (**ser_out**) nebo příjem (**ser_in**) je nutné nejdříve nastavit parametry kanálu (**ser_init**) a prioritu vysílače (**ser_oip**) nebo přijímače (**ser_iip**). Při aktivním RT jádru nepoužívat prioritu 1!

Dále je možné definovat funkci volanou při ukončení vysílání (**ser_oef**) nebo příjmu (**ser_ief**). U příjmu lze také definovat seznam startznaků (**ser_stch**) a stopznaků (**ser_ench**).

Uvedení nulového ukazatele na obslužnou funkci znamená, že funkce nebude volána vůbec. Nulový ukazatel na seznam start- nebo stop- znaků znamená, že se nebude kontrolovat.

Signál RTS je pro kanály 1 a 2 řízen automaticky (dáno zapojením). Má-li být signál RTS využíván i pro kanál 0, je nutno ho řídit pomocí funkce **ser_rts**.

```
int ser_init(int k, long b, int f)
```

Nastaví sériový kanál **k** (0 až 2) na rychlost **b** v Baudech. Dle použitého krystalu jsou možné pouze některé rychlosti (pro 14,75MHz jsou dostupné všechny standardní rychlosti 300, 600, 1200, 2400,4800, 9600, 19200, 38400, 57600, 115200). Zakáže přerušení od sériového kanálu. Parametr **f** určuje datový formát (0=8bit bez parity). K dispozici jsou předdefinované konstanty:

```
#define parity_none 0x00
#define parity_odd 0x20
#define parity_even 0x60
#define use_cts 0x80
```

```
int ser_oip(int k, int p)
```

Nastaví prioritu přerušení **p** (0 až 6) vysílacího kanálu **k** (0 až 2). Uplatní se vždy až při volání funkce **ser_out**.

```
int ser_oef(int k, void (*o)(void*,long), void *p)
```

Nastaví obslužnou funkci **o** volanou při ukončení vysílání kanálu **k** (0 až 2). Pro **o=0** se nic nevolá. Obslužné funkci jsou jako parametry předány obecný ukazatel **p** a počet znaků, které se nepodařilo úspěšně odeslat.

```
int ser_echo(int k, int s)
```

Zapne (**s=1**) nebo vypne (**s=0**) kontrolu příposlechu při vysílání kanálu **k** (0 až 2). Po aktivaci se připoslouchává každý odeslaný znak a v případě neshody je vysílání přerušeno. Použitelné hlavně pro RS-485.

**int ser_out(int k, char *odkud, long kolik)**

Přesměruje vektor od vysílače a odešle souvislý blok znaků z adresy **odkud** o délce **kolik** kanálem **k** (0 až 2). Pro **odkud=0** přenos pokračuje následujícím znakem za posledně odeslaným.

Pozor při odesílání dat uložených v lokálních proměnných! Odesílaný blok musí existovat nejméně do odeslání posledního znaku, zatímco lokální proměnné zanikají při opuštění funkce. Problém je možné řešit použitím deklarace *static*, ale funkce potom není reentrantní.

**int ser_rts0(int stav)**

Řízení signálu RTS0. Parametr **stav**=logická hodnota signálu $\overline{\text{RTS0}}$ (negované RTS0). Funkce vrací předchozí stav $\overline{\text{RTS0}}$.

**int ser_ostat(int k)**

Vrátí stav vysílacího kanálu **k** (0 až 2). Funkce vrací hodnotu 0 (FREE), pokud je kanál volný, nebo 1 (BUSY), pokud vysílá.

**long ser_olen(int k)**

Vrátí počet zatím neodeslaných znaků z kanálu **k** (0 až 2).

**int ser_iip(int k, int p)**

Nastaví prioritu přerušení **p** (0 až 6) přijímacího kanálu **k** (0 až 2). Uplatní se vždy až při volání funkce **ser_in**.

**int ser_ief(int k,void(*o)(void*,long),void *p)**

Nastaví **o** jako obslužnou funkci volanou po ukončení příjmu z kanálu **k** (0 až 2). Pro **o=0** se nevolá. Příjem je ukončen buď po přijetí určeného počtu znaků nebo po detekci meziznakové mezery delší než určené pomocí funkce **ser_in**. Obslužné funkci je předán obecný ukazatel **p** a celkový počet přijatých znaků.

**int ser_wef(int k, void (*hotovo)(void *p), void *p)**

Nastaví obslužnou funkci ukončení čekání, totožné s **ser_ief**, ale nepředává se počet přijatých znaků (obsah EBX není definován).

**int ser_stch(int k,char *s, long d)**

Aktivuje seznam startznaků pro kanál **k** (0 až 2). Parametr **s** ukazuje na řetězec požadovaných startznaků o délce **d**. Pro **s=0** se na startznak nečeká. Tato funkce způsobí, že následné volání **ser_in** stejného kanálu bude přijímané znaky ignorovat, dokud nepřijme znak ze seznamu. Přijetím některého ze startznaků se tato kontrola ihned vypne a v případě nového příjmu je nutné volat tuto funkci znovu.

**int ser_ench(int k,char *s, long d)**

Aktivuje seznam stopznaků pro kanál **k** (0 až 2). Parametr **s** ukazuje na řetězec požadovaných stopznaků o délce **d**. Pro **s=0** se na stoptznak nečeká. Tato funkce způsobí, že následné volání **ser_in** stejného kanálu ukončí příjem ihned při přijetí některého znaku ze seznamu. Seznam stopznaků zůstane nadále v platnosti.

**int ser_flush(int k)**

Vyprázdní vstupní registr sériového kanálu **k** a vynuluje příznaky chyb. Je vhodné volat před zahájením příjmu funkcí **ser_in**.

**int ser_in(int k, byte *kam, long maxdel, int timeout)**

Přijme blok bajtů z kanálu **k** (0 až 2) na adresu **kam**. Parametr **timeout** udává časový interval ve znacích s následujícím významem dle hodnoty:

- **t>0** povolená meziznaková mezera,
- **t<-1** časový limit na celou zprávu,
- **t=0** ponechat časovač beze změny,
- **t=-1** zastavit časovač.

Příjem bude ukončen po splnění některé z podmínek:

- přijetí **maxdel** znaků,
- vypršení timeoutu,
- přijetím stopznaku.

Vynuluje příznaky chyb příjmu.

**int ser_wt(int k, int wtime)**

Zablokuje přijímací kanál **k** (0 až 2) na dobu **wtime** znaků.

**int ser_istat(int k)**

Vrátí stav přijímacího kanálu **k** (0 až 2). Funkce vrací hodnotu 0 (FREE), pokud je kanál volný, nebo 1 (BUSY), pokud přijímá.

**int ser_err(int k)**

Vynuluje příznaky chyb příjmu. Vrátí jejich stav před vynulováním. Význam jednotlivých bitů:

- bit 0 .... *framing error* – vadný stopbit (chyba přenosu),
- bit 1 .... *parity error* – špatná parita (chyba přenosu),
- bit 2 .... *overrun error* – zahlcení přijímače (asi neměl dostatečnou prioritu).

**long ser_ilen(int k)**

Vrátí počet přijatých znaků kanálu **k** (0 až 2).

**int ser_lock(int k)**

Zamkne kanál pro výhradní použití aktuálním procesem.Tato funkce by se měla volat před prvním voláním **ser_init** v procesu.

**int ser_unlock(int k)**

Uvolní kanál zamknutý pro výhradní použití. Pokud nebyl zamknut v tomto procesu, tak nemůže být ani uvolněn.

### 2.5.6 Řízení časovačů

K dispozici je osm osmibitových čítačů, které je možné spojit na čtyři šestnáctibitové. Velikost se volí adresou kanálu (parametr **c**, 0 až 7 jsou samostatné osmibitové, **0x10**, **0x32**, **0x54** a **0x76** jsou šestnáctibitové páry). Kanály 0 a 1 jsou normálně využity pro udržování systémového čítače. Některé další kanály mohou být dočasně využity systémem při aktivaci některých funkcí (např. pro obsluhu sériových kanálů, viz. popis konkrétních služeb).

**int cnt_lock(int c)**

Uzamkne časovač **c** pro výhradní použití aktuálním procesem.

**int cnt_unlock(int c)**

Uvolní časovač **c** pro použití ostatními procesy.

**int cnt_init(int c, int i, void(*obsl)(void))**

Zastaví časovač **c**, přiřadí mu prioritu přerušení **i** a obslužnou funkci **obsl**.

**int cnt_start(int c, int p)**

Spustí časovač **c** s nastavenou periodou **p** v mikrosekundách. Přesnost nastavení a dosažitelný rozsah závisí na řídícím krystalu procesoru a použitém čítači (liché kanály dovolují nastavit větší dělící poměr pro prescaler).

**int cnt_stop(int c)**

Zastaví časovač **c**. Je možné zastavit i pouze jeden ze spuštěného šestnáctibitového páru.

**int cnt_stat(int c)**

Zjistí stav časovače **c**. Vrací 0, pokud je v klidu, nebo 1, pokud čítá nebo je využíván jinou službou.

### 2.5.7 Ovládání obvodu reálného času

Funkce, jejichž název začíná **rtc**... přistupují přímo do obvodu RTC a během jejich provádění je zakázáno přerušení na dobu až dvou milisekund. Ostatní funkce využívají systémový čítač v paměti inkrementovaný 1024× za sekundu. Inkrementování zajišťuje obsluha INT1 s prioritou 1. Pro kontrolu a synchronizaci jsou využity časovače 0 (s prioritou 1) a 1(s prioritou 6). Obsluha časovače 0 zajišťuje korekci systémového čítače každých 250 ms. V případě odstavení INT1 na dobu

500 ms nebo více dojde k aktivaci přerušení od časovače 1 a jeho obsluha zajistí synchronizaci podle času v obvodu RTC.

`void tick(tword *t)`

Zapíše do `t` hodnotu ze systémového časovače (počet 1/1024 sekundy od 00:00 1.1.1980).

`unsigned long time(void)`

Vrátí počet sekund od 00:00 1.1.1980.

`long get_time(void)`

Vrátí aktuální čas jako packed BCD číslo ve tvaru: `hhmmss0d`, kde `hh`=hodiny (0÷23), `mm`=minuty (0÷59), `ss`=sekundy (0÷59), `d`=den v týdnu (0..neděle, 1..pondělí, ..., 6..sobota).

`long get_date(void)`

Vrátí aktuální datum jako packed BCD číslo ve tvaru: `rrrrmmdd`, kde `rrrr`=rok (1980÷2099), `mm`=měsíc (1÷12), `dd`=den (1÷31).

`void set_timeout(tword *p, long t)`

Nastaví čas pro timeout. Parametr `p` je ukazatel na místo pro uložení hodnoty času (šest bajtů), `t` je požadovaná doba timeoutu v 1/1024 s.

`int timeout(tword *p)`

Po vypršení timeoutu vrací hodnotu 1, jinak 0.

`long rtcdate(void)`

Přečte aktuální datum. Funkce vrací hexadecimální číslo ve tvaru `rrrrmmdd`.

`long rtctime(void)`

Přečte aktuální čas. Funkce vrací hexadecimální číslo ve tvaru `hhmmss0d`.

`void rtcinit(long d, long t)`

Nastaví čas a datum. Parametr `d`=datum ve tvaru `rrrrmmdd`, `t`=čas ve tvaru `hhmmssDD` (`DD`..den v týdnu). Zároveň aktualizuje systémový čítač v paměti.

## 2.6 Jádro reálného času TORTOS16

### 2.6.1 Přehled

Plánovač procesů umožňuje po své inicializaci spuštění až patnácti procesů (celkem šestnácti vláken: 15 procesů + pozadí) v preemptivním multitaskingu (v rámci možností, protože procesorové jádro TMP900H pracuje trvale v režimu SYSTEM, může kterýkoliv proces zákazem přerušení přepínání procesů zastavit). Priorita všech procesů je fixní a shodná s PID. Pozadí má PID=0 (nejnižší priorita). Všechny časy jsou odvozovány z výstupu obvodu RTC s frekvencí 1024 Hz, tedy přibližně po 1 ms. Sám plánovač spotřebuje podle rychlosti procesoru maximálně 4 až 7 procent strojového času. Plánovač procesů využívá přerušení INT1 spouštěné výstupem RTC s kmitočtem 1024 Hz.

### 2.6.2 Funkce pro řízení procesů

`int spawn(int pid, void(*pep)(void), long *stk, int spt, int rit)`

Spustí proces `pid`, `pep`=vstupní bod procesu, `stk`=vrchol zásobníku procesu (nejvyšší adresa zásobníku), `spt`=spouštěcí perioda, `rit`=doba běhu. Funkce vrací 0, pokud proběhla v pořádku, jinak vrací bitovou masku indikující spuštěné procesy, popř. číslo 1, indikující chybné vstupní parametry.

`int sleep(int pid, int tp)`

Pozastaví provádění procesu `pid` na dobu `tp`. Pro `tp`=0 pokračuje jako obvykle při příštím vyvolání.

`int stop(int pid)`

Zastaví provádění procesu `pid`.

`int wakeup(int pid)`

Obnoví provádění procesu `pid`.

`int prqerr(int m)`

Zjistí/vynuluje příznaky chyb spouštění procesů. Parametr `m` je maska pro vynulování (funkcí AND). Vrátí původní stav příznaků.

`void wait(long t)`

Čeká dobu `t`. Vhodné hlavně pro pozadí, proces je stále aktivní (není to sleep!).

`int get_pid(void)`

Zjistí `pid` aktuálního procesu.

## 2.7 Defaultní nastavení

### 2.7.1 Přehled

Datová struktura uložená ve vyhrazené oblasti ve Flash EPROM. Určuje chování systému po RESETu. Pro modifikaci je nutné pořídit kopii do RAM (`get_defaults`), provést potřebné úpravy a opětovně uložit do Flash EPROM (`put_defaults`).

```
typedef struct {
  char startup_delay; //doba vyčkávání po
                        RESETu v [s]
  char dbg_con;       //kanál pro znakový výstup,
                        -1 = žádný
  long dbg_baudů;     //rychlost pro znakový
                        výstup
}BIOS_defaults;
```

`int get_defsize(void)`

Vrátí velikost paměti potřebné ke stažení defaultních nastavení. Obvykle služba vyžaduje velikost paměti odpovídající nejmenšímu bloku paměti Flash EPROM (tedy 256B).

`void get_defaults(void *d)`

Na adresu `d` zkopíruje defaultní nastavení.

`int put_defaults(void *d)`

Zapíše defaultní nastavení z adresy `d` do Flash EPROM. Vrací 0, pokud zápis proběhl úspěšně.

## 2.8 Monitor

### 2.8.1 Přehled

Vestavěný monitor umožňuje nahrávat do paměti RAM a Flash EPROM a spouštět kód na libovolné adrese.

Příjem příkazů pro monitor (nahrátí programu apod.) probíhá přes kanál 0 (RS-232) rychlostí 38400 Bd. Všechny příkazy monitoru končí znakem „mezera“ nebo menším a větším než 0. Jako potvrzení vrací znak „hvězdička“ nebo svoji specifickou odpověď ukončenou `\r\n`. Při chybě syntaxe/rozsahu apod. vrací `?\r\n`. Interpretace řetězce skončí na prvním nepatřičném znaku. Přijetí kteréhokoliv příkazu v době vyčkávání po RESETu zablokuje automatické spuštění uživatelského programu nebo UDE debuggeru.

### 2.8.2 Příkazy monitoru

`I000000` – vrací infořetězec.

`Raaaaaa,dd` – read memory, vrací `=nnnn...nn`.

`Waaaaaa<nn` – write memory.

`Ydddddd<llllll,ssssss` – copy memory (destination<length,source).

`Mn=ss,mm,bb:jjjjjj` – MSARn=`ss`, MAMRn=`mm`, BnCS=`bb`, skočí na `jjjjjj`,
pro `j=0` ... `ret`,
pro `j=0xffff00` ... `jp (j)` (tabulka vektorů).

`S2ddaaaaaannnn...nnss` – Motorola S-format 24bit do `fbuf`, nedovolí zapsat do posledních 64 kB (BIOS).

`S804000000FB` – flush `fbuf` před „`*`“ může vrátit:
„`%`“ ... O.K.,
„`!`“ ... chyba programování FEPROM.

**SUE** – aktivuje odemykací sekvenci před každým zápisem **fbuf** (FEPROM).

**SUD** – zakáže odemykací sekvenci před každým zápisem **fbuf** (RAM, implicitní).

**SSnnnn** – definuje velikost **fbuf** v bajtech (implicitně se určuje automaticky podle typu osazené FEPROM).

**Gxwa,xbc,xde,nnnn,...,nnnn:aaaaaa** – **ld regs**, **push nnnn**, **call aaaaaa**, první tři parametry musí být uvedeny jako 32bit, vrátí „**xhl\r\n**“, po návratu bude obnoven XSP, XIX a XIZ!

**G:aaaaaa** – verze bez parametrů.

**T:RAM** – provede kompletní destruktivní(!) test RAM, vrátí O.K. nebo **!badadr**.

**T:RTC** – test RTC, vrátí **d:rrrrmmdd t:hhmmss00**.

## 2.9 Dodatky

### 2.9.1 Využití registrových bank

Architekturu programu silně ovlivní to, zda bude nebo nebude určen pro běh v multitaskingu.

Při prvním vyvolání funkce **spawn()** se aktivuje plánovač úloh, který využívá registrovou banku 2. Procesům je při spuštění vždy přidělena banka 0. Banky 1 a 3 jsou volně k použití uživatelským programem.

Pro multitáskové úlohy je velmi důležité důsledné používání zámků zajišťujících výhradní použití prostředků.

### 2.9.2 Volací konvence

Při kompilaci programů napsaných v jazyce C pro procesory Toshiba založené na jádru TLCS-900 nabízí UDE-compiler dva způsoby pro předávání parametrů při volání funkcí.

Při prvním, klasickém, se všechny parametry předávají na zásobníku a to tak, že první parametr je nejblíže k vrcholu zásobníku, hned pod návratovou adresou. Parametry ze zásobníku odstraňuje volající. Vstupní bod takovéto funkce označuje návěští tvořené jménem funkce, kterému předchází znak „podtržítko“. Jako implicitní tento způsob se zvolí buďto řádkovým parametrem -Xc nebo ve zdrojovém textu řádkem:

```
#pragma cdecl
```

V deklaraci funkce lze použít modifikátor **__cdecl**:

```
int __cdecl funkce(...);
```

Při druhém způsobu se funkci předávají parametry v registrech. První tři parametry jsou po řadě v registrech (nebo jejich částech, pokud jsou kratší než 32bitů) XWA, XBC, XDE. Případné další parametry jsou předány na zásobníku tak, že nejblíže vrcholu je poslední parametr, těsně pod návratovou adresou. Za odstranění parametrů ze zásobníku je zodpovědná volaná funkce. Vstupní bod takovéto funkce označuje návěští tvořené jménem funkce, kterému předchází znak tečka. Jako implicitní tento způsob se zvolí buďto řádkovým parametrem -Xa nebo ve zdrojovém textu řádkem:

```
#pragma adecl
```

V deklaraci funkce je možné použít modifikátor **__adecl**:

```
int __adecl funkce(...);
```

Předávání parametrů v registrech je rychlejší i úspornější na délku kódu a prostor na zásobníku (pro funkce s nejvýše třemi parametry se tak ukládá pouze návratová adresa).

Funkce mající parametry typu double je vhodnější překládat s modifikátorem **__cdecl**.

V obou případech se návratová hodnota (je-li nějaká) předává v registru XHL (nebo jeho části, pokud je kratší než 32bitů).

### 2.9.3 Volatile

Při deklaraci proměnných sloužících např. jako semafory je nutno důsledně používat klíčové slovo **volatile**. V rámci optimalizace je překladač schopen vygenerovat naprosto nefunkční kód. Příklad:

```
int semafx;              //deklarace semaforu
void udalost(){          //obsluha události
  ...
  semafx=1;              //nastaví semafor
};

main(){
  ...
  semafx=0;
  ...
  while(semafx!=1);      //čekání na událost
  ...
};
```

V tomto případě přeloží kompilátor čekací smyčku **while** zhruba takto:

```
          ...
          ld wa,(_semafx)
 s000:    cp wa,1
          jr nz,s000
          ...
```

Takže pokud nebyl semafor nastaven již před vstupem do smyčky, tak program tuto smyčku nikdy neopustí. Pokud se proměnná **semafx** nadeklaruje takto:

```
volatile int semafx;
```

pak překladač vygeneruje toto:

```
          ...
 s000:    cpw (_semafx),1
          jr nz,s000
          ...
```

Teď již smyčka opravdu testuje aktuální stav semaforu.

## 2.10 Debug SWI2

Po RESETu vypíše BIOS hlášení přes kanál 1 (RS-485) rychlostí 19200 Bd. Použitý kanál lze změnit modifikací parametru **dbg_con** v **BIOS_defaults**. Na tento kanál také směřují diagnostická hlášení i jednoduchý debug aktivovaný přes SWI2. Po aktivaci vypíše obsahy registrů a výzvu „**?**“ (otazník). Potom akceptuje jednoduché povely (všechno velká písmena, čísla v hex):

**Raaaaaa,dd** – zobrazí **dd** bajtů paměti od adresy **aaaaaa**.

**Waaaaaa,dd=nn,nn,...,nn** – zapíše **dd** bajtů do paměti od adresy **aaaaaa**.

**Xaaaaaa** – provede instrukci **call aaaaaa**.

Při stisku nekorektní klávesy vypíše „**?**“. Stisk mezerníku ukončí obsluhu SWI2.