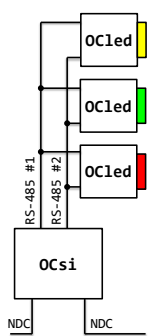


Odpovědi pište na zvláštní odpovědní list s vaším jménem a fotografií. Pokud budete odevzdávat více než jeden list s řešením, tak se na 2. a další listy nezapomeňte podepsat. Do zápatí všech listů vždy napište i/N (kde i je číslo listu, N je celkový počet odevzdaných listů).

### Otázka č. 1

Byli jsme vybráni firmou *Starmon* jako subdodavatel části jejich inovativního železničního zabezpečovacího zařízení *SIRIUS* – naším úkolem bude vyrobit modul LED návěstní svítilny železničního návěstidla. Každé návěstidlo má v systému *SIRIUS* ve své patě instalován počítačový modul OCsi, který je připojený na staniční počítačovou síť NDC. Z modulu OCsi vedou dvě nezávislé multidrop sériové sběrnice RS-485 (na obou je jejich jediným masterem) – je to sběrnice podobná RS-232, ale navíc s podporou pro více



slave zařízení a využívající diferenciální přenos. Přenos dat na RS-485 probíhá po jednotlivých **9-ti** bitových bytech. V každé návěstní svítelně bude nainstalován jeden náš modul OC1ed jehož součástí je matice vysoce svítivých LED diod jedné barvy. Každý z modulů OC1ed v nějakém návěstidle je jako slave připojen nezávisle na obě RS-485 sběrnice od návěstního modulu OCsi – viz obrázek vlevo.

Na obou sběrnících se používá následující komunikační protokol: každý slave má na sběrnici přiřazenu unikátní 8-bitovou adresu odpovídající funkci návěstní svítilny (žlutá = 1, zelená = 2, červená = 4); byty s nastaveným nejvyšším bitem (bit 8) jsou adresové, tj. spodních 8 bitů = adresa cílového slava; byty s vynulovaným nejvyšším bitem (bit 8) jsou datové byty, tj. spodních 8 bitů = nějaký příkaz pro zařízení, jehož adresa byla přítomna v posledním přijatém adresovém bytu – **pozor**: po jednom adresovém bytu může následovat libovolné množství datových bytů, které pak všechny náležejí stejnému zařízení. Pokud má nějaká návěstní svítilna svítit, tak jejímu modulu OC1ed (na jeho adresu) posílá modul OCsi každých 40 ms datový byte s hodnotou \$55 po celou dobu očekávaného svícení. Pokud má být svítilna zhaslá, tak se na její adresu žádá data neposílají. V konkrétním stavu návěstidla může být požadavek na svícení více svítilen současně. Svítilna ovládaná modulem OC1ed má svítit právě tehdy, pokud v posledních 80 ms přišel **po každé** z RS-485 sběrnic na jeho adresu **alespoň jeden** datový byte \$55.

Naprogramujte v Pascalu chování modulu OC1ed pro *červené světlo* jako firmware pro níže popsaný 32-bitový MCU, který v modulu OC1ed používáme. Procesor má podporu pro 8 zdrojů přerušení (číslované 0-7), tabulka vektorů přerušení začíná na adrese \$FF000000 a každá položka je jedna 32-bitová adresa. V  $\mu\text{C}$  jsou integrované dva nezávislé RS-485 řadiče, kde každý z nich je v modulu OC1ed připojen na jednu ze dvou RS-485 sběrnic návěstidla. Každý z řadičů má jeden 16-bitový read-only paměťově mapovaný registr (1. řadič na adrese \$E0000000, 2. řadič na adrese \$E0000100), jehož spodních 9 bitů obsahuje hodnotu posledního bytu přijatého po dané sběrnici (**obsah registru se jeho čtením nemění**). Při příjmu libovolného 9-bitového bytu je řadičem vygenerována žádost o přerušení (ukončení žádosti je provedeno automaticky ve spolupráci s jádrem procesoru při volání interrupt handleru) – 1. řadič

generuje IRQ 1, 2. řadič generuje IRQ 2.  $\mu\text{C}$  v sobě navíc obsahuje časovač, který každou milisekundu generuje IRQ 0. Na adrese \$E0000200 je paměťově mapovaný 8-bit write-only registr GPIO řadiče s 8 digitálními výstupními linkami, kde na každou z těchto linek je připojena jedna osmina LED diod z diodové matice na modulu OC1ed. Při hodnotě 1 v bitu registru GPIO řadiče konkrétní osmina LED diod svítí. Při běžném chování OC1ed modulu má celá diodová matice svítit nebo být celá zhasnutá.

### Otázka č. 2

Nakreslete časový diagram přenosu dvou bytů 0xFF 0x5F po sériové sběrnici RS-232 (bez hodinového signálu), pokud se používá přenos s 8-bitovými byty, 1 start bitem, 1 stop bitem, a přenosová rychlost je 2400 baud. Detailně vysvětlete, jak se pozná začátek a konec přenosu, a jak přijímající rozpozná, že mu posíláme právě data 0xFF 0x5F.

### Otázka č. 3

Popište a detailně vysvětlete důvod vzniku alespoň 3 typických faults/exceptions, které mohou synchronně vznikat při provádění kódu v nějakém moderním mikroprocesoru.

### Otázka č. 4

Obecně v nějaké aplikaci může dojít ke stavu, kdy chybnou přílišnou rekurzí v programu dojde k přepsání kódu programu obsahem jeho lokálních proměnných. Jak se tato situace nazývá? Bylo by možné tomu v nějakém operačním systému běžícím na nějakém moderním procesoru (kde by dokázal běžet např. OS Linux nebo OS Windows 7) zabránit? Tedy že by chyba v programu byla detekována dříve, než začne procesor provádět „data lokálních proměnných“ jako kód? Detailně vysvětlete proč, případně jak.

### Otázka č. 5

Předpokládejme, že v jádře operačním systému poskytujícím podporu pro vícevláknové zpracování chceme v Pascalu naimplementovat API proceduru Sleep(s : Longint), která způsobí, že vlákno, které ji zavolá, nebude ve zpracování dalších instrukcí pokračovat dříve než za s sekund. Takovou operaci lze implementovat několika způsoby – vyberte pro zmíněný OS se souběžným během mnoha aplikací ten nejvhodnější, a v Pascalu popište implementaci procedury Sleep a všech ostatních částí OS, které budou pro její fungování potřeba (soustředte se jen na části související se samotným procesem od začátku do konce čekání volajícího vlákna). Můžete počítat s tím, že každá cílová počítačová platforma vám poskytuje všechna běžná a pro vás důležitá zařízení a řadiče.

### Otázka č. 6

Popište a vysvětlete, jakým způsobem se typicky překládají a spouštějí programy napsané v Javě nebo v jazyce C#. Do svého vysvětlení zahrňte, co to je, a jaký význam a výhody má v tomto kontextu tzv. *intermediate language*.

**Společná část pro otázky označené X**

Předpokládejte, že máme počítač se zjednodušenou variantou 32-bitového **big-endian** procesoru *Motorola 68000*. Tento procesor má následující **registry**:

8 obecných registrů D0 až D7 – lze je použít pouze jako přímou zdrojovou nebo cílovou hodnotu nějaké operace; 8 obecných tzv. adresových registrů A0 až A7 – pro jejich zápis je potřeba použít speciální instrukce, v běžných instrukcích je lze použít pouze jako operand typu *adresa*.

**Registr A7 se běžně používá jako stack pointer**

(předpokládejte typickou organizaci volacího zásobníku). Dále má procesor 32-bitový registr PC a 16-bitový registr stavu CCR (obsahující všechny běžné příznaky).

**Instrukční sada:** Většina instrukcí má 32-bit (přípona .l v assembleru), 16-bit (přípona .w), i 8-bit (přípona .b) variantu dané operace. Procesor má mimo jiné následující instrukce (<op> = libovolná varianta operandu, viz dále, An = libovolný z registrů A0 až A7, Dn = libovolný z registrů D0 až D7, **cílový** je vždy **nejpravější** operand):

Instr.	Operandy	Velikost operandů	Popis
MOVE	<op>, <op>	8, 16, 32	kopie hodnoty zdr→cíl
MOVEA	<op>, An	32	kopie do adr. registru
ADD	Dn, <op> <op>, Dn	8, 16, 32	přičtení datového nebo k datovému reg.
ADDA	<op>, An	32	přičtení k adr. registru
JSR	<op>	32	volání podprogramu
RTS	žádné	žádné	návrat z podprogramu

**Operandy:** Za <op> je možno dosadit libovolnou z následujících variant operandů (zapisováno v syntaxi běžného Motorola 68000 assembleru):

- #imm hodnota immediate
- Dn operace s obsahem registru Dn
- (An) operace s pamětí daná obsahem registru An
- imm(An) operace s pamětí, cílová adresa je daná součtem obsahu registru An a hodnoty imm
- -(An) tzv. predekrementace: jako součást instrukce je hodnota v registru An zmenšena o velikost prováděné operace v bytech, a nová hodnota v registru An slouží jako cílová adresa
- (An)+ tzv. postinkrementace: aktuální hodnota registru An slouží jako cílová adresa, po provedení instrukce je ale hodnota v registru An automaticky zvětšena o velikost provedené operace v bytech

**Příklad programu:** Pokud je v registru A0 hodnota 0x00100000, a v registru D1 hodnota 0xFFFFFFFF, a od adresy 0x00100000 jsou v paměti následující byty:

```
00 00 00 05 00 00 00 00 00 00 03, potom po:
move.l #7, (a0)+      { nastavení 32-bit hodnoty 7 do 32-bit
                      hodnoty na adrese dané reg. A0, a zvětšení obsahu A0 o 4 }
add.w 6(a0), d1      { zvětšení spodních 16-bitů registru D1
                      o 16-bitovou hodnotu na adrese A0 + 6 }
adda.l #4, a0        { zvětšení adresy v A0 o 4 }
```

bude v registru A0 hodnota 0x00100008, v D1 bude 0xFFFF0002, a v paměti od adresy 0x00100000 bude: 00 00 00 07 00 00 00 00 00 00 03

**Volací konvence:** Při volání procedur a funkcí se parametry typicky předávají na volacím zásobníku zprava doleva, 32b, 16b, i 8b návratové hodnoty se vracejí v registru D0.

**Otázka č. 7 (X)**

Níže uvedený text reprezentuje strojový kód části programu nahraného v počítači od adresy 0x00001000 (zde začíná hlavní program). Kód jsme disassemblovali do standardního assembleru Motorola 68000 (viz vždy pravá část řádku po bytech strojového kódu dané instrukce). Zapište, jak by asi v Pascalu mohl vypadat kód původního programu a všech uvedených procedur a funkcí. Pro všechny procedury a funkce zmíněné v kódu (i ty, jejichž implementaci neznáte) zvolte nějaké názvy a ty v kódu konzistentně používejte.

```
00001000 4E B9 00 00 10 36 jsr    #$00001036
00001006 3F 00                move.w d0, -(a7)
00001008 4E B9 00 00 10 36 jsr    #$00001036
0000100E 3F 00                move.w d0, -(a7)
00001010 4E B9 00 00 10 2A jsr    #$0000102A
00001016 58 8F                adda.l #4, a7
00001018 2F 3C 00 00 00 01 move.l #1, -(a7)
0000101E 2F 00                move.l d0, -(a7)
00001020 4E B9 00 00 10 42 jsr    #$00001042
00001026 50 8F                adda.l #8, a7
00001028 4E 75                rts
0000102A 30 2F 00 04        move.w 4(a7), d0
0000102E 32 2F 00 06        move.w 6(a7), d1
00001032 D0 81                add.l d1, d0
00001034 4E 75                rts
```

**Otázka č. 8 (X)**

Předpokládejte, že bychom chtěli dát na instrukci na adrese \$101E breakpoint – tedy, že se má laděný program zastavit právě před provedením této instrukce. Jak debugger breakpoint do programu vloží? Změní se nějak paměť obsazená laděným programem? Pokud ano, tak jak, pokud ne, tak proč.

**Otázka č. 9 (X)**

Předpokládejte, že níže uvedený program budeme překládat pro CPU Motorola 68000. Do místa {\*} napište Pascal kód, který za běhu přepíše proceduru Print tak, aby nezavolala WriteLn, ale rovnou se vrátila do hlavního prog.:  
**procedure Print; begin WriteLn('Hello'); end; begin {\*} Print; end.**

**Otázka č. 10**

Naprogramujte v Pascalu funkci Conv s níže uvedeným prototypem, která převede **číslo ve floating-point formátu single** (typ *single* je 32-bit číslo dle IEEE 754, tj. mantisa je normalizována se skrytou 1 a zabírá spodních 23 bitů, pak následuje 8-bit exponent uložený ve formátu bias +127, poslední bit, tedy MSb, je znaménkový) **do floating-point formátu double** (typ *double* je 64-bit dle IEEE 754, tj. mantisa je norm. se skrytou 1 a zabírá spodních 52 bitů, pak následuje 11-bitový bias +1023 exponent, a poslední bit je znaménkový). Celý výpočet/zpracování zapište jen s využitím celočíselné aritmetiky Pascalu (bez použití Pascal typů pro reálná čísla), a zvažte možnost použít pro výpočet bitové operace podporované v Pascalu. Poznámka: typ *qword* je unsigned 64-bit integer, *longword* je 32-bitový.

**function** Conv(flt32 : longword) : qword;