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).

Společná část pro otázky označené X

Předpokládejte, že jsme si pomocí debuggeru zobrazili hex view následující části fyzického adresového prostoru (navíc předpokládejte, že pracujeme na systému s 32-bitovým fyzickým adresovým prostorem a s 32-bitovým **little-endian** procesorem, kde logické adresy rovnou odpovídají adresám fyzickým):

```
0x028CF53C  48 44 39 72 0f 00 00 00  HD9r....
0x028CF544  ff 00 fa 78 56 00 00 5f  ˙.úxV.._
0x028CF54C  c2 a7 91 03 43 70 79 00  Â§‘.Cpy.
0x028CF554  00 00 00 00 04 17 39 72  ......9r
```

Otázka č. 1 (X)

Napište ve Free Pascalu bez chyb přeložitelný program, který bude každé 4 byty souvisle uložené v adresovém prostoru mezi adresami `0x028CF53C` až `0x028CF55B` interpretovat jako floating-point číslo typu `single`, a pro každé takové číslo na samostatný řádek na standardní výstup vypíše: v šestnáctkové soustavě počáteční a koncovou adresu, na které číslo leží, a hodnotu reálného čísla v desítkové soustavě, které data v paměti reprezentují. Pro uvedená data bude tedy výpis obsahovat 29 řádků textu a bude vypadat následujícím způsobem:

```
028CF53C-028CF53F: 3.669585522E+30
028CF53D-028CF540: 1.194255015E-29
...
028CF558-028CF55F: 3.666083264E+30
```

Otázka č. 2 (X)

Napište v desítkové soustavě reálné číslo, které program z otázky 1 vypíše, když vypisuje interpretaci dat ležících na adresách `0x028CF549` až `0x028CF54C`. Předpokládejte, že typ `single` je 32-bitové floating-point číslo dle standardu IEEE 754, tj. mantisa je normalizována se skrytou 1 a zabírá spodních 23 bitů, pak následuje 8-bitový exponent uložený ve formátu s posunem [bias] +127, a poslední bit, tedy `MSb`, je znaménkový bit.

Otázka č. 3 (X)

Předpokládejte, že v uvedeném počítači je jednoduchá paralelní 32-bitová systémová sběrnice se sdílenými adresovými a datovými vodiči a s 32-bitovým adresovým prostorem (pokud vám to pomůže, tak si můžete představit např. sběrnici PCI – pro tuto otázku bude ale dostačující i minimalistická single master systémová sběrnice navržená dle typické jednoduché paměťové sběrnice). Jaké všechny signály/vodiče taková sběrnice minimálně musí mít? U každého signálu vysvětlete jeho význam.

Otázka č. 4 (X)

Předpokládejte, že při běhu programu z otázky 1 bude procesor určitě někdy zpracovávat instrukci `load` pro 32-bitový operand začínající na adrese `0x028CF549`. V průběhu zpracování takové instrukce bude jistě procesor potřebovat načíst 4 byty od adresy `0x028CF549`. Nakreslete kompletní časový diagram všech přenosů na systémové sběrnici z otázky 3, které budou potřeba pro dokončení čtení uvedených 4 bytů (pokud znáte koncept procesorové cache, tak předpokládejte, že CPU žádnou cache nemá). Předpokládejte, že procesor podporuje nezarovnaná čtení, ale paměťový subsystém vyžaduje čtení a zápisy zarovnané na 4 byty.

Otázka č. 5

Na počítačích IBM PC kompatibilních je součástí firmware i API funkce poskytující možnost přečtení souvislé skupiny sektorů z pevného disku – pro zavolání funkce je vyvolat softwarové přerušení `INT 13h`, pro které je třeba předem do registru `AH` připravit hodnotu `02h`, do registrů `ES:BX` logickou adresu pro uložení dat sektorů, do registru `AL` počet sektorů, a do registru `CX` číslo prvního sektoru ze skupiny. Dále víme, že tabulka vektorů přerušení je během bootování uložena v RAM od adresy `0000h:0000h`, a kód firmware v nejhornější části adresového prostoru.
Pro tento počítač máme boot loader nějakého jednoduchého OS, který pro načtení jádra OS používá výše uvedenou funkci `02h` přerušení `13h` tak, že si napevno vždy nechává načíst `128` sektorů počínaje sektorem `1`, kde očekává, že je jádro uložené. Pokud bychom disk nyní rozdělili na oddíly, a uvedený boot loader bychom nahráli do boot sektoru oddílu, který bude začínat na `4096`. sektoru disku (tj. jádro OS bude nyní uložené ve `128` sektorech od sektoru `4097`), bylo by možné naprogramovat vhodný boot manager, který by umožnil nechat původní kód boot loaderu beze změny? Co bude muset takový boot manager uložený v `0`. sektoru disku udělat, aby zajistil správné fungování původního boot loaderu? Sektory `1` až `129` jsou zabrané dalším oddílem, jehož data se samozřejmě během bootování nesmí modifikovat.

Otázka č. 6

Vysvětlete, jaký je u sériové linky RS-232 význam tzv. *stop bitu*. Dále vysvětlete, zda by dávalo smysl mít u RS-232 variantu komunikačního protokolu bez stop bitu? Pokud by varianta bez stop bitu byla v něčem problematická, tak popište příklad situace (přenosu dat), kde by k nějakým problémům mohlo dojít.

Otázka č. 7

Předpokládejte, že v jádře jednoduchého OS s kooperativním přepínáním vláken chceme v Pascalu pro 32-bitový CPU naprogramovat vylepšenou variantu API procedury `CreateThread` s níže uvedenou deklarací. Procedura má v kontextu volajícího procesu vytvořit nové vlákno a zajistit, že při prvním naplánování nového vlákna začne běžet kód jeho hlavní procedury `threadProc` s argumentem `arg` nastaveným na hodnotu `param`. Napište implementaci `CreateThread` co nejpřesněji (stačí na úrovni detailu z přednášky). Argumenty se ukládají na volací zásobník zprava doleva a odstraňuje je volající.

```
type
  PThreadProc = procedure(arg : longword);
procedure CreateThread(
  threadProc : PThreadProc; param : longword);
```

Otázka č. 8

Mikroprocesor Motorola 6800 je **8-bitový little-endian** CPU s akumulátorovou architekturou a dvěma nezávislými 8-bitovými registry akumulátoru, a s **16-bitovým adresovým** prostorem. Procesor má **8-bitové registry**: akumulátor `A`, akumulátor `B`, a příznakový registr s běžnými příznaky. Dále má **16-bitové registry**: pomocný indexový registr `X`, registr `SP` (ukazuje na první volné místo na zásobníku) a registr `PC` (Program Counter).

V instrukční sadě jsou mimo jiné následující instrukce (za `?` lze dosadit `A` nebo `B` pro výběr akumulátoru, se kterým bude instrukce pracovat):

- `LDA?` (load accumulator), `STA?` (store accumulator)
- `PSH?` (push accumulator), `PUL?` (pull/pop accumulator)
- `ADD?` (add, nastavuje carry), `ADC?` (add with carry)
- `LDX` (load X register), `STX` (store X register)

Všechny výše uvedené instrukce mají v assembleru 1 explicitní operand v jedné z následujících variant:

- 8-bitový immediate: `#$xx`
- 16-bitový immediate (jen pro registr X): `#$xxxx`
- absolutní adresa: `$xxxx`
- indexovaná adresa (adresa X + delta): `X,delta`

Aritmetické instrukce modifikují příznakový registr standardním způsobem. Dle přenesené hodnoty modifikují příznakový registr i instrukce `LDA?`, `STA?`, `LDX`, a `STX`.

Kromě již uvedených instrukcí má procesor ještě instrukce:

- `TSX` (transfer **SP+1** to `X`), `TXS` (transfer **X–1** to `SP`)
- `BEQ` (branch if equal)
- `BRA` (branch always, tj. nepodmíněný skok)
- `RTS` (return from subroutine, tj. ekvivalent `RET`)

Napište v Pascalu bez použití inline assembleru kód procedury (i s deklarací), která by mohla být běžným překladačem přeložena do níže uvedeného kódu v assembleru 6800 (předpokládejte, že procedura používá běžnou Cčkovou volací konvenci, tj. argumenty se předávají na volacím zásobníku zprava doleva, a odebírá je volající):

```
label1:                    LDAA X,3
   TSX                     ADCA #$00
   LDX  X,2                STAA X,3
   LDAA X,0                LDAA X,4
   BEQ label2              ADDA #$01
   TSX                     STAA X,4
   LDX  X,4                LDAA X,5
   STAA X,0                ADCA #$00
   TSX                     STAA X,5
   LDAA X,2                BRA  label1
   ADDA #$01            label2:
   STAA X,2                RTS
```

Otázka č. 9

Předpokládejte, že programujeme DLL knihovnu pro práci s 8-portovým GPIO řadičem Texas Instruments PCF8574 připojeným na I^2C sběrnici (datasheet je přílohou tohoto zadání – pro tuto otázku jsou důležité hlavně části 8.3 a 8.4). Komunikaci po I^2C sběrnici ale neprogramujeme sami, nýbrž očekáváme, že při volání naší funkce dostaneme od volajícího záznam, který popisuje existenci procedur pro čtení a zápis na I^2C sběrnici – konkrétní jména ani umístnění těchto procedur ale při překladu naší knihovny neznáme, můžeme ale předpokládat, že budou mít následující parametry (poslední dva argumenty vždy popisují velikost a pozici bufferu se samotnými datovými byty, které se mají po I^2C poslat, resp. kam mají být samotné datové byty načteny), a že každé jejich volání vždy provede 1 kompletní I^2C zápisovou, resp. čtecí transakci.

```
type
  PByte = ^byte;
procedure I2cWrite(slaveAddress : byte;
  bytesToWrite : byte; bufferBaseAddr : PByte);
procedure I2cRead(slaveAddress : byte;
  bytesToRead : byte; bufferBaseAddr : PByte);
```

Vaším úkolem je napsat ve Free Pascalu deklaraci záznamu `TI2cDriverProcInfo`, který má popisovat existenci dvou procedur s výše uvedenými prototypy, a pak naprogramovat funkci:

```
function GpioGetInputAt(
  i2cDriver : TI2cDriverProcInfo; pin : byte) : byte;
```

která s využitím předaných procedur pro zápis/čtení přes I^2C vrátí stav konkrétního GPIO vstupu číslo `pin` (`0` až `7`) na GPIO řadiči PCF8574 (předpokládejte, že vždy budeme komunikovat s řadičem, který má piny `A0` až `A2` připojené na zem).

Otázka č. 10

Předpokládejte následující deklarace (kde typ `longword` je 32-bitový celočíselný bezznaménkový, typ `word` je 16-bitový celočíselný bezznaménkový):

```
type
  PLongword = ^longword;
  PWord = ^word;
procedure Conv(src : PLongword; dst : PWord);
```

Napište ve Free Pascalu implementaci procedury `Conv` tak, aby převedla vstupní textový null-terminated řetězec `src` z kódování UTF-32 LE do kódování UTF-16 LE a výsledné znaky UTF-16 LE null-terminated řetězce uložila do paměti na místo, kam ukazuje argument `dst` (předpokládejte, že váš kód poběží pouze na little-endian platformách, a že na místě, kam ukazuje proměnná `dst`, je dostatek nevyužité paměti).

Unicode znaky z rozsahu `$010000` až `$10FFFF` se v UTF-16 kódují následujícím způsobem:

(1) Od kódu znaku se odečte hodnota `$010000`, a výsledné 20-bitové číslo se rozdělí na dvě 10-bitové části, které se zakódují dle následujících pravidel.

(2) Nejvyšších 10-bitů 20-bitové hodnoty se uloží do nejnižších 10 bitů prvního 16-bitového surrogate znaku (leží na nižší adrese). Horních 6 bitů první surrogate má být nastaveno na (vlevo je hodnota bitu 15, vpravo bitu 10): `1101 10`

(3) Nejnižších 10-bitů 20-bitové hodnoty se uloží do nejnižších 10 bitů druhého 16-bitového surrogate znaku (leží na vyšší adrese). Horních 6 bitů druhé surrogate má být nastaveno na (od první surrogate se liší pouze 10. bitem): `1101 11`