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

Předpokládejte, že do níže uvedené globální proměnné `buffer` v našem programu pro ornitology načteme nekomprimovaný jednokanálový (mono) zvukový záznam hlasu pěnkavy obecné:

```
var
  buffer : array[0..88199] of byte;
```

Kolik sekund zvuku se nám do proměnné vejde? Pokud byste potřebovali znát nějaké další charakteristiky zaznamenaného zvuku, tak si je vhodně zvolte, a uveďte je ve svém řešení. Do své odpovědi zahrňte vysvětlení termínů *sample* a *sample rate* v kontextu proměnné `buffer`.

Otázka č. 2

Předpokládejte standardní 32-bitovou variantu PCI sběrnice. Vysvětlete, jaký je význam signálů `IRDY#` (*initiator ready*) a `TRDY#` (*target ready*). Kdo dané signály generuje a kdo je čte? Uveďte a vysvětlete také příklad nějaké situace, kde jsou signály `IRDY#` a `TRDY#` potřebné a bez nich by přenos po sběrnici nemohl v pořádku proběhnout.

Otázka č. 3

Některé verze operačního systému Linux používají na 32-bitové architektuře IA-32 pro vstup do kernelu instrukci `INT 80h`. Vysvětlete, co daná instrukce dělá, a proč se v Linuxu pro volání API funkcí kernelu v aplikacích nepoužívá standardní instrukce `CALL` místo instrukce `INT`.

Otázka č. 4

Naprogramujte v Pascalu funkci `Conv` s níže uvedeným prototypem, která převede **nenulové** číslo v bezznaménkovém fixed-point formátu `8+24` do floating-point formátu *single* (při převodu byste měli zachovat co největší přesnost; zvažte využití bitových operací pro provedení převodu; jako součást výpočtu můžete použít i nějaký algoritmus lineární vzhledem k počtu bitů vstupu/výsledku) – výsledná hodnota ve formátu *single* je návratovou hodnotou funkce (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):

```
function Conv(fixed : longword) : longword;
```

Otázka č. 5

Předpokládejte, že máme 16 bytové SRAM paměťové moduly, které používají 4 bitová slova. Dva takové moduly bychom chtěli připojit na 8-bitovou paměťovou paralelní sběrnici s 64 bytovým adresovým prostorem (s oddělenými datovými a adresovými vodiči). Paměť modulů má být přístupná ve spodních 32 bytech adresového prostoru. Nakreslete a okomentujte obrázek vhodného zapojení těchto modulů včetně běžných kontrolních vodičů a případných pomocných zařízení (u nich vysvětlete jejich fci).

Otázka č. 6

Předpokládejte, že máme počítač s variantou 32-bitového procesoru Intel 80486 běžícího v 32-bitovém režimu s vypnutou podporou pro stránkování (virtuální i fyzický adresový prostor procesoru má šířku 32-bitů, virtuální/logická adresa se přímo rovná adrese fyzické). Procesor má obecnou registrovou architekturu, a mimo jiné 7 obecných 32-bitových registrů `EAX`, `EBX`, `ECX`, `EDX`, `ESI`, `EDI`, `EBP`, dále má 32-bitový registr `ESP` (stack pointer), 32-bitový registr `EIP` (program counter), a příznakový registr s běžnými příznaky. ISA obsahuje instrukce `MOV`, `PUSH`, `POP`, `ADD` (sčítání bez přenosu), `SHL`, `CMP`, `JC` (jump if carry), `JNC` (jump if not carry), a `RET` s běžnou sémantikou.
Víme, že od adresy `001616A0h` je v paměti počítače uložen kód funkce `DoSomeMagic` bez argumentů, která vrací 32-bitovou unsigned hodnotu. Funkce používá variantu Cčkové volací konvence, kdy je návratová hodnota ukládána do registru `EAX`.
Dále máme k dispozici disassembler generující kód ve variantě Intel syntaxe, tedy: cílový operand instrukce je vždy nejvíce vlevo; hodnota v hranatých závorkách znamená variantu instrukce s operandem typu adresa, tj. např. `[x]` znamená hodnotu operandu na adrese `x`.
Po spuštění disassembleru na adresu `001616A0h` jsme zjistili, že kód funkce `DoSomeMagic` je následující:

```
001616A0 55                 push ebp
001616BE A1 38 91 16 00     mov  eax,[00169138h]
001616C3 83 C0 05           add  eax,5
001616C6 83 F8 07           cmp  eax,7
001616C9 73 19              jnc  01616E4h
001616CB A1 3C 91 16 00     mov  eax,[0016913Ch]
001616D0 D1 E0              shl  eax,1
001616D2 3B 05 38 91 16 00  cmp  eax,[00169138h]
001616D8 76 0A              jc   01616E4h
001616DA A1 38 91 16 00     mov  eax,[00169138h]
001616DF 83 C0 01           add  eax,1
001616E2 EB 08              jmp  01616ECh
001616E4 A1 3C 91 16 00     mov  eax,[0016913Ch]
001616E9 83 C0 01           add  eax,1
001616EC 5D                 pop  ebp
001616ED C3                 ret
```

Zapište v Pascalu její kód i její kompletní deklaraci bez použití inline assembleru. Jména globálních proměnných, která nejsou z disassemblovaného kódu zřejmá, si vhodně zvolte. Předpokládejte, že typ `longword` je 32-bitové celé číslo bez znaménka.

Otázka č. 7

Uveďte příklad několika (alespoň tří) nejdůležitějších API funkcí, které bude poskytovat firmware běžného osobního počítače. Pro každou takovou funkci/proceduru napište její hlavičku v Pascalu (tj. jméno a seznam argumentů a návratovou hodnotu včetně jejich typů) a krátký komentář k její funkci. Také odpovězte na otázku: kdo a kdy bude tyto funkce využívat?

Otázka č. 8

Předpokládejte, že programujeme část operačního systému pro procesorovou platformu se 4 KiB stránkami, s jednoúrovňovou stránkovací tabulkou, a následujícím formátem 32-bitových položek stránkovací tabulky (který je shodný s formátem použitým u procesorů Intel Pentium):

31:12	11:9	8	7	6	5	4	3	2	1	0
Frame	I	G	I	D	A	PCD	PWT	U/S#	R/W#	P

kde jednotlivé bity mají následující význam:

- P = `present` (pokud je `0`, jsou bity 1 až 31 procesorem ignorovány [v našem OS si v takové situaci do nich ukládáme různé pomocné informace, a žádný z těchto bitů pak nemá původní význam, viz níže]),
- R/W# = `Read-only/Read-write`,
- U/S# = `User/Supervisor`,
- `I` = `Ignored` (hodnota těchto bitů je CPU ignorována),
- `Frame` = číslo rámce přiděleného této stránce,
- G, D, A, PCD, PWT – žádný z těchto bitů nevyužíváme.

Vaším úkolem je v Pascalu naprogramovat následující funkci `CanRead`, která v argumentu `ptBase` dostane bázovou adresu stránkovací tabulky nějakého procesu nějaké běžné aplikace, a v argumentu `vaddr` nějakou virtuální adresu v adresovém prostoru stejného procesu. Funkce má vrátit `true`, pokud bude moci kód aplikace běžící v kontextu tohoto procesu při stávajícím stavu stránkovací tabulky přečíst 1 byte z adresy `vaddr`, `false` vrací v opačném případě (k funkci napište krátký komentář vysvětlující její chování).

```
type
  PLongword = ^longword;

function CanRead(
  ptBase : PLongword; vaddr : longword
) : boolean;
```

Společná část pro otázky označené X

Předpokládejte, že máme akcelerometr ADXL345 od společnosti Analog Devices, který komunikuje po I²C sběrnici – na sběrnici má napevno nastavenou adresu `0x1D`. Akcelerometr máme nakonfigurovaný na měření zrychlení ve všech 3 osách v plném 11-bitovém rozlišení v rozsahu ±4 g. Akcelerometr má mimo jiné 6 jednobytových read-only registrů (v závorce je vždy uvedena adresa registru v adresovém prostoru akcelerometru) [informace o čipu a obrázek uvedený níže jsou převzaty z datasheetu výrobce – viz http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf]: DATAX0 (`0x32`), DATAX1 (`0x33`), DATAY0 (`0x34`), DATAY1 (`0x35`), DATAZ0 (`0x36`), DATAZ1 (`0x37`). Registry *X* obsahují 11-bitové znaménkové číslo ve dvojkovém doplňku reprezentující naměřené zrychlení ve směru osy X, podobně registry *Y* pro osu Y, a *Z* pro osu Z. Pro každou z os vždy registr DATA*1 obsahuje MSB 11-bitové hodnoty (tj. bity 7 až `0` obsahují bity `10` až 3 naměřené hodnoty), registr DATA*0 obsahuje v nejvyšších 3 bitech (7 až 5) nejnižší 3 bity naměřené hodnoty (2 až `0`). Akcelerometr podporuje příkaz čtení jednoho bytu z registru ve svém adresovém prostoru (viz single-byte read), i příkaz pro burst čtení více bytů z registrů ve svém adresovém prostoru (viz multi-byte read). V případě operace multi-byte read se předává adresa registru na nejnižší adrese z přenášených dat (jako u běžné burst operace). Viz obrázek dole na stránce.

Otázka č. 9 (X)

Předpokládejte, že v Pascalu programujete ovladač akcelerometru ADXL345 pro OS s preemptivním přepínáním vláken. Kromě vašeho ovladače bude v adresovém prostoru jádra nahrán i ovladač řadiče I²C sběrnice, jehož následující API fce máte k dispozici, a i je ve svém řešení využijte:

```
procedure SendStop;
procedure SendStart;
function ReceiveByte : byte;
function SendByte(b : byte) : boolean;
procedure SendAckOrNack(ack : boolean);
```

Všechny uvedené funkce uvnitř v případě potřeby čekají pasivně, čili není problém je použít synchronně v sekvenčním algoritmu. Funkce `SendByte` vrací `true`, pokud byl po odeslaném bytu přijmut ACK, a vrací `false` při příjmu NACK – stejnou sémantiku má i argument `ack` procedury `SendAckOrNack`. Pokud jste zavolali proceduru `SendStart`, tak dokud nezavoláte `SendStop`, tak ovladač I²C mezi jednotlivými voláními svých API funkcí drží sběrnici ve wait stavu (udržuje signál SCL v nule). Čekání na idle stav sběrnice, stejně tak jako detekci ztráty arbitrace a restart transakce řeší ovladač I²C řadiče sám za vás. Formát I²C adresového bytu je pro 7-bitové adresy následující:

MSB							LSB
							R/W#
slave address							

Naprogramujte následující API funkci ovladače ADXL345:

```
function ReadXAxis : longint;
```

která vrací naměřené zrychlení v ose X ve stejném rozsahu jako datové registry akcelerometru, ale jako 32-bitové znaménkové číslo `longint`. Pokud dojde k chybě v komunikaci s akcelerometrem (je např. odpojen), tak má funkce vrátit hodnotu `0`. Funkce by neměla volající vlákno zablokovat aktivním čekáním, ale může volající vlákno uvést do stavu pasivního čekání.

Otázka č. 10 (X)

Na dole uvedeném obrázku označuje poznámka 1 u STARTu po přenosu adresy registru možnost, že akcelerometr v tomto místě podporuje jak repeated start, tak rozdělení přenosu na dvě transakce, tj. že v místě označeného STARTu může být též odeslána STOP condition a po opětovném odeslání START condition může být dokončen přenos ve druhé I²C transakci. Detailně vysvětlete, zda má jedna nebo druhá varianta nějaké výhody pro mastera transakce.