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

Navrhněte a popište co nejjednodušší formát zvukových souborů vhodných pro ukládání nekomprimovaného *mono* i *stereo* zvuku v libovolné běžně používané kvalitě. Popište hlavičku takového souboru ve formě Pascalového záznamu, a vysvětlete, co jednotlivé jeho složky znamenají.

Společná část pro otázky označené X

Předpokládejte, že máme počítač, kde je systémovou sběrnicí varianta paralelní 32-bitové sběrnice PCI s taktovací frekvencí 33 MHz. Sběrnice podporuje 32-bitový paměťový adresový prostor a 16-bitový I/O adresový prostor. Na základní desce je řadič paměti, který podporuje připojení maximálně 2 GB operační paměti v modulech DRAM. Nejvyšší 1 MB paměťového adresového prostoru je obsazen pamětí EEPROM s firmware počítače. V počítači je použita varianta CPU vycházející z architektury procesorů Intel 80386. Předpokládejte, že na tomto počítači nabootujeme operační systém MS-DOS – veškeré aplikace v něm spuštěné běží pod tzv. reálným režimem CPU, kdy se 80386 chová jako původní CPU Intel 8086 (a má v tomto režimu i stejnou instrukční sadu) – Intel 8086 **16-bitový little-endian** CPU s podporou 20-bitového fyzického adresového prostoru. Logické adresy jsou 32-bitové, kde spodních 16 bitů logické adresy je tzv. `offset`, a horních 16 bitů je tzv. `segment` – přepočet z logické na fyzickou adresu je následující: `fyzická = segment * 16 + offset` (tedy 1 fyzická adresa je ekvivalentně reprezentována mnoha různými logickými adresami). U našeho CPU 80386 je v reálném režimu spodních 20 bitů fyzické adresy získáno stejně jako u 8086, horních 12 bitů je vždy nulových.

Otázka č. 2 (X)

Pro použití v uvedeném počítači navrhněte HCI zvukové karty, která má být schopná přehrávat *mono* i *stereo* zvuk v libovolné běžně používané kvalitě – nicméně předpokládejte maximální velikost vzorku (sample) 24-bitů. Všechny registry zvukové karty přístupné z CPU mají být striktně memory mapped, mají být dostupné i v reálném režimu procesoru, a nevadí nám, když zakryjí část v počítači nainstalované paměti DRAM nebo EEPROM s firmware. Adresy registrů mají být součástí HCI a tedy je nějak vhodně zvolte. Zvuková karta nemusí pro přenos dat používat DMA.

Otázka č. 3 (X)

Naprogramujte v Pascalu funkci s níže uvedenou deklarací:

```
type PByte = ^byte;
function PhysToLog(a : longword) : PByte;
```

která jako argument získá 20-bitovou fyzickou adresu platnou na uvedeném počítači, a která má vrátit libovolný ukazatel takový, abychom jeho dereferencí získali přístup právě k předané fyzické adrese. Tedy např. zápis hodnoty `$15` na fyzickou adresu `$60` bychom pak mohli provést jako:

```
PhysToLog($60)^ := $15;
```

Otázka č. 4 (X)

Napište v Pascalu program, který ze souboru, jehož jméno je předáno jako argument na příkazové řádce programu při jeho spuštění (argument z příkazové řádky lze ve formě textového řetězce získat voláním standardní Pascalové funkce `ParamStr(1)`), přečte záznam zvuku a přehraje ho na zvukové kartě. Předpokládejte, že soubor má právě takový formát, jaký jste si navrhli v řešení **otázky č. 1**, a že zvuková karta má právě takové HCI, jaké jste si navrhli v řešení **otázky č. 2**. Pro přístup k registrům zvukové karty využijte funkci `PhysToLog` z **otázky č. 3**. Předpokládejte, že zvuková karta má dostatečně velký vnitřní buffer, do kterého se vejde kompletní záznam zvuku z libovolného vám předaného souboru, navíc si i v programu můžete alokovat dostatečně velké pole pro uložení celého vstupního souboru.

Otázka č. 5

Předpokládejte, že programujeme DLL knihovnu pro práci s EEPROM pamětí NXP Semiconductors PCA24S08A připojenou na I^2C sběrnici (datasheet je přílohou tohoto zadání – pro tuto otázku jsou důležité hlavně části 6.1 a 6.3). 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. Pokud je argument `sendStop` nastavený na `true`, tak se na konci transakce vyšle *Stop* bit a sběrnice přejde do idle stavu; pokud je argument `false`, tak *Stop* bit není vyslán a pomocí clock stretchingu je sběrnice držena v zabraném stavu do zavolání dalšího příkazu `I2cWrite` nebo `I2cRead`.

```
type PByte = ^byte;
procedure I2cWrite(
  slaveAddress : byte; sendStop : boolean;
  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 EepromReadByte(
  i2cDriver : TI2cDriverProcInfo; byteAddr : word
) : byte;
```

která s využitím předaných procedur pro zápis/čtení přes I^2C přečte a vrátí hodnotu bytu uloženého na adrese `byteAddr` v připojené paměti EEPROM.

Otázka č. 6

Vysvětlete hlavní rozdíly mezi paměťmi typu EEPROM a DRAM.

Otázka č. 7

Upravte kód následující procedury `Sort` tak, aby na dvou procesorovém systému pod běžným moderním OS s preemptivním přepínáním vláken mohla běžet netriviálně rychleji než na systému jednoprocesorovém:

```
type
  PLongint = ^longint;
procedure Quicksort( { nepoužívá globální proměnné }
  pole : PLongint; n : longint); forward;
procedure Merge(     { nepoužívá globální proměnné }
  zdroj1 : PLongint; zdroj2 : PLongint;
  n1, n2 : longint; cil : PLongint); forward;

{ n >= 1 }
procedure Sort(
  zdroj : PLongint; cil : PLongint; n : longint
);
var
  pul : longint;
begin
  pul := n div 2;
  Quicksort(zdroj, pul);
  Quicksort(zdroj + pul, n - pul);
  Merge(zdroj, zdroj + pul,
        pul, n - pul, cil);
end;
```

Napište deklarace všech syscallů (procedur a funkcí), které budete požadovat od jádra OS. Ke každé napište stručný popis jejího očekávaného chování.

Společná část pro otázky označené Y

Předpokládejte, že jsme CPU 80386 z otázek X přepnuli do tzv. chráněného režimu (protected mode 32), ve kterém používá jinou instrukční sadu než v otázkách X. V tomto režimu se CPU chová jako **32-bitový little-endian** procesor s obecnou registrovou architekturou a s 32-bitovým logickým i fyzickým adresovým prostorem (logická adresa se nyní přímo rovná adrese fyzické – tedy předpokládejte, že se **nevyužívá stránkování**). Procesor má obecné registry EAX, EBX, ECX, EDX, DSI, EDI, EBP, 32-bitový příznakový registr EFLAGS s běžnými příznaky, registr ESP (stack pointer, ukazuje na poslední využitý byte, roste dolů), a registr EIP (instruction pointer). V instrukční sadě jsou **instrukce pro standardní bitové operace** a dále mimo jiné **i následující instrukce** (příznakový registr modifikují pouze aritmetické operace, ale instrukce přenosu dat nikoliv):

- `MOV reg,imm32/[addr]` (load register)
- `MOV [addr],reg` (store register)
- `MOV reg0,reg1` (transfer from `reg1` to `reg0`)
- `ADD reg,imm32/[addr]/reg` (add without carry)
- `SUB reg,imm32/[addr]/reg` (subtract without carry)
- `PUSH imm32/[addr]/reg` (32-bit push)
- `POP [addr]/reg` (32-bit pop)
- `JMP addr` (direct jump), `JNZ addr` (jump if not zero)
- `CMP reg,[addr]` (32-bit compare)
- `CALL addr` (direct call), `CALL [addr]` (indirect call)
- `RET` (return from subroutine)

Všechny výše uvedené instrukce i bitové operace se dvěma operandy mají vždy vlevo cílový a vpravo zdrojový operand. Instrukce mohu mít jednu z následujících variant operandů (povolené varianty viz definice konkrétní instrukce):

- 32-bitový immediate: `imm32`
- absolutní adresa `[addr]`, kde `[addr]` může být jedno z:
  - `[imm]` adresa daná konstantou
  - `[reg +/- imm]` adresa daná součtem/rozdílem obsahu registru `reg` a konstanty `imm`
- libovolný registr: `reg`

Otázka č. 8 (Y)

Napište v Pascalu bez použití inline assembleru kód funkce (i s deklarací), která by mohla být běžným překladačem přeložena do níže uvedeného kódu, který jsme z paměti disassemblovali do assembleru 80386 (víme, že funkce používá běžnou Cčkovou volací konvenci, tj. argumenty se předávají na volacím zásobníku zprava doleva, a odebírá je volající, návratová hodnota se vrací v registru `EAX`):

```
00401941 55              push   ebp
00401942 89 e5           mov    ebp,esp
00401944 83 ec 04        sub    esp,0x00000004
00401947 8b 45 10        mov    eax,[ebp+0x10]
0040194a 3b 45 0c        cmp    eax,[ebp+0x0c]
0040194d 75 08           jnz    0x00401957
0040194f 8b 45 08        mov    eax,[ebp+0x08]
00401952 89 45 fc        mov    [ebp-0x04],eax
00401955 eb 1b           jmp    0x00401972
00401957 8b 45 10        mov    eax,[ebp+0x10]
0040195a 83 c0 01        add    eax,0x00000001
0040195d 50              push   eax
0040195e ff 75 0c        push   [ebp+0x0c]
00401961 ff 75 08        push   [ebp+0x08]
00401964 e8 d8 ff ff ff  call   0x00401941
00401969 83 c4 0c        add    esp,0xc
0040196c 03 45 08        add    eax,[ebp+0x08]
0040196f 89 45 fc        mov    [ebp-0x04],eax
00401972 8b 45 fc        mov    eax,[ebp-0x04]
00401978 89 ec           mov    esp,ebp
0040197a 5d              pop    ebp
0040197b c3              ret
```

Otázka č. 9 (Y)

Předpokládejte, že se nám spolu s výše uvedeným kódem podařilo do adresového prostoru nahrát i kód a data aplikace debuggeru s grafickým uživatelským rozhraním podobným editoru Lazarus. S pomocí tohoto debuggeru jsme umístili breakpoint na první řádek výše uvedené funkce. Jak přesně debugger zařídí, že pokud dojde k zavolání uvedené funkce, tak se program doopravdy „zastaví“, a bude čekat, dokud v aplikaci debuggeru nestiskneme klávesu F9 pro pokračování běhu programu? Co přesně se děje při dosažení breakpointu, a co při stisku klávesy F9?

Otázka č. 10 (Y)

Předpokládejte, že výše uvedený kód je součástí programu nahraného od bázové adresy `$00401900`. Pokud bereme v potaz pouze výše uvedený strojový kód, bylo by možné celý program zcela beze změny nahrát i na bázovou adresu `$2F801900`? Detailně vysvětlete proč, resp. co by případně bylo nutné v programu upravit. Změnila by se nějak vaše odpověď při zapnutém stránkování?