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

Pro nejběžnější variantu PCI sběrnice nakreslete časový diagram průběhu hodnot na všech důležitých vodičích při úspěšném provedení kompletního zarovnaného čtení jednoho slova s hodnotou `$C0000000` z adresy `$F0000000` (transakce byla iniciována prováděním instrukce `mov eax, [$F0000000]` v CPU, kde po provedení instrukce bude v registru `eax` hodnota `$C0000000`. Nakreslete průběh pro každý vodič sběrnice PCI zvlášť, přičemž seskupit do jednoho smíte jen vodiče, které mají v celém průběhu transakce shodnou hodnotu. K vašemu obrázku napište krátkou legendu, tj. pro každý použitý vodič vysvětlete jeho význam, a uveďte, které zařízení v uvedené konfiguraci „generuje“ signál na kterém vodiči.

Otázka č. 2

Předpokládejte, že v typickém OS máme ovladač zařízení A (např. AHCI řadič disků) přímo připojeného na sběrnici PCI Express, a jiný ovladač zařízení B (např. externí USB disk) přímo připojeného na sběrnici USB. Liší se nějak principiálně způsob komunikace se zařízením A a se zařízením B z pohledu jejich ovladačů, když jsou obě sběrnice PCIe i USB sériové a data se po nich přenášejí ve formě paketů? Vysvětlete proč!

Otázka č. 3

Popište, jaké všechny podmínky musí být splněny, aby mohlo dojít ve vícevláknové aplikaci k deadlocku. Jako příklad napište kód tří vláken takové aplikace, a to s následujícími vlastnostmi: pokud libovolná dvě z těchto vláken doběhnou dříve, než začne běžet třetí, tak k deadlocku nikdy nedojde. Pokud všechna tři vlákna poběží dostatečně „současně“, tak při vhodném naplánování k deadlocku dojít může.

Otázka č. 4

Předpokládejme, že v operačním systému poskytujícím podporu pro vícevláknové zpracování chceme v Pascalu naimplementovat 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 (soustřeďte 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. 5

Zapište v šestnáctkové soustavě hodnotu 32-bitové proměnné `i` typu `Longint` po provedení níže uvedeného kódu v Pascalu, když víme, že se kód bude překládat pro little endian platformu:

```
i := -16; i := ((i shr 2) and $AA)
                xor ((1 shl 7) or 129);
```

Otázka č. 6

Úloha: Čistě v assembleru níže uvedeného 32-bitového procesoru naimplementujte kompletní níže uvedenou funkci `f` při použití varianty Cčkové volací konvence (parametry se předávají na zásobníku zprava doleva, parametry odstraňuje volající, návratová hodnota se vrací v registru EAX) a za předpokladu, že `Longword` je celočíselný bezznaménkový 32-bitový typ:

```
function f(a : Longword) : Longword;
begin
   if a = 0 then begin
      f := 1;
   end else begin
      f := a * f(a – 1);
   end;
end;
```

Instrukční sada: Kód implementujte pro počítač s variantou 32-bitového procesoru Intel 80386 běžícího v 32-bitovém režimu s vypnutou podporou pro stránkování (virtuální i fyzický adresový prostor procesoru má šířku 32-bitů, virtuální adresa se přímo rovná adrese fyzické). Procesor má obecnou registrovou architekturu, a mimo jiné 7 obecných 32-bitových registrů `EAX`, `EBX`, `ECX`, `EDX`, `EBP`, `EDI`, `ESI`, dále má 32-bitový registr `ESP` (stack pointer), a 32-bitový registr `EIP` (program counter), a příznakový registr se všemi běžnými příznaky. Můžete využít instrukce `MOV`, `ADD`, `SUB` (odečítání), `MUL` (násobení), `CMP`, `JZ`, `JNZ`, `JMP`, `PUSH`, `POP`, `CALL`, `RET`, které mají všechny standardní chování. Instrukce ovlivňují příznaky standardním způsobem, aritmetické instrukce jsou dvouoperandové. Pro zápis vašeho kódu využijte běžné Intel syntaxe, tedy: cílový operand instrukce je vždy nejvíce vlevo; immediate hodnota je libovolné číslo bez prefixu; hodnota v hranatých závorkách znamená variantu instrukce s operandem typu adresa, tj. např. `[x]` znamená hodnotu operandu na adrese `x`; běžné instrukce procesoru 80386 mají i variantu, kde se adresa operandu spočítá jednoduchým výpočtem, a tedy např. zápis `[x+y]` znamená operand ležící na adrese, která vznikne součtem hodnot `x` a `y`, kde `x` i `y` mohou být libovolný registr, pouze `y` může být i hodnota immediate, u instrukcí s více operandy smí být pouze jeden adresa v `[]`. Cíl skoku si můžete v assembleru označit jako label podobně jako v Pascalu, tedy libovolné jméno s dvojtečkou.

Př.: pokud je v registru `EDI` hodnota `$00AA0050`:

```
něco:                { následující instrukce mov je cílem skoku }
   mov eax, [edi+4]  { načtení 4B hodnoty z adresy $00AA0054
                       a její uložení do registru EAX }
   jmp něco          { proveď skok na adresu instrukce za
                       lablem něco, tj. zde na instrukci mov }
```

Otázka č. 7

Předpokládejte počítač s plně 16-bitovým procesorem Intel 8086 taktovaným na frekvenci 8 MHz a připojeným na 16-bit variantu 8 MHz sběrnice ISA (20-bit adresový prostor, přenos jednoho slova za 6 taktů). Procesor má 6 bytovou prefetch queue, a výpočetní jednotka procesoru (ALU) pracuje zcela paralelně s načítáním do prefetch queue. Všechny jednotky sdílejí jedno rozhraní k systémové sběrnici, a výpočetní jednotka má vyšší prioritu při potřebě přístupu k ní. V dokumentaci procesoru jsme našli následující tabulku o základním časování instrukcí v ALU:

Instruction	Instruction Operands	Clock Cycles
ADD/SUB	`reg,reg`	3
ADD/SUB	`reg,mem`	9
ADD/SUB	`mem,reg`	16
MOV	`mem,reg`	9

K systémové sběrnici je připojena paměť EPROM namapovaná na adresu `4600:0000`. V řádkovém debuggeru pod MS-DOSem jsme zjistili následující obsah této paměti (procesor má 4 obecné registry AX, BX, CX, DX, cílový je vždy nejlevější operand, operand v závorkách `[x]` znamená čtení/zápis dat z/na adresy/u x):

```
-u 4600:0010
4600:0010 01D8      ADD     AX,BX
4600:0012 01C8      ADD     AX,CX
4600:0014 2B060010  SUB     AX,[1000]
4600:0018 A30010    MOV     [1000],AX
4600:001B E9E201    JMP     0200
```

Předpokládejte, že procesor bude po zapnutí napájení provádět kód v této paměti ROM. Po provedení počátečních inicializací je v kódu proveden skok na adresu `4600:0010` (pozor: každý skok na procesoru Intel 8086 vede ke kompletnímu vyprázdnění prefetch queue). V tomto scénáři určete čas v µs mezi dokončením 2. a 3. instrukce (tj. doba zpracování 3. instrukce, tj. instrukce `SUB AX, [1000]`). Hodnotu můžete uvést ve formě zlomku. Zapište (nakreslete) a detailně vysvětlete i postup výpočtu!

Otázka č. 8

Následující obrázek obsahuje část screenshotu hex editoru, který zobrazuje obsah 91 bytů dlouhého binárního souboru:

```
    0001 0203 0405 0607 0809 0A0B 0C0D 0E0F
00  127D 016C 0075 0074 00FD 0020 006B 006F
10  0148 01F6 FFFF FFF6 6E86 1BF0 F921 0940
20  0000 AE40 0A00 0000 0A48 0065 006C 006C
30  006F 0000 0000 0000 C015 4000 2480 4400
40  0000 00F0 FFEF 40DD CCBB AA80 FF7F 47EF
50  BEAD DE04 0657 006F 0077 00
```

Víme, že všechna data jsou v souboru uložena jako little endian, a že od 63. bytu (počítáno od 0) je v souboru uloženo 64-bitové reálné číslo s pohyblivou desetinnou čárkou ve formátu IEEE 754 double, tj. mantisa je normalizována se skrytou 1 a zabírá spodních 52 bitů, pak následuje 11-bitový exponent uložený ve formátu s posunem (bias) +1023 a 1 znaménkový bit. Zapište hodnotu tohoto reálného čísla v desítkové soustavě.

Otázka č. 9

Napište program v jazyce Pascal, který na standardní výstup (tj. pomocí procedury `WriteLn`) vypíše text „`Little Endian`“ nebo „`Big Endian`“ bez uvozovek podle toho, na jaké platformě bude spuštěn (resp. pro kterou bude přeložen). Připomenutí: prefixový unární operátor `@` slouží v Pascalu pro získání adresy libovolné proměnné.

Otázka č. 10

Předpokládejte, že chcete programovat větší množství různých aplikací, které všechny budou používat grafické uživatelské rozhraní (GUI). Každou z naprogramovaných aplikací budete chtít v binární spustitelné podobě distribuovat pro operační systémy Linux a Mac OS X. Oba tyto operační systémy mají ale rozdílné API (navzájem nekompatibilní) pro tvorbu grafického uživatelského rozhraní, a zároveň programovací jazyk, který budete používat pro programování aplikací, nemá žádnou podporu pro tvorbu GUI. Aplikace chcete programovat tak, aby byly přenositelné na úrovni zdrojového kódu mezi oběma uvedenými systémy. Jakým způsobem to nejlépe zařídíte? Jakým způsobem pak bude probíhat překlad veškerého potřebného kódu každé takové aplikace, až do stavu, kdy máme finální spustitelné soubory?