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 máme počítač s původní 8-bitovou sběrnicí ISA (tj. podpora pro 20-bitový paměťový adresový prostor, a 16-bitový I/O adresový prostor). Sběrnice je paralelní, nemá podporu pro burst přenosy a je taktována na 8 MHz.

Dále předpokládejte, že chceme pro tuto sběrnici navrhnout část řadiče sítové karty pro připojení k 10BASE-T síti. Navrhovaná část bude zodpovědná pouze za příjem paketů (zcela nezávislou část pro odesílání paketů budeme řešit později). Řadič bude mít v sobě zabudován 12144 B buffer implementovaný pamětí SRAM, který bude sloužit jako cyklická fronta pro příjem Ethernetových rámců (maximální délka rámce 1518 bytů). Při příjmu každého kompletního paketu má řadič vyvolat žádost o přerušení. Dále má řadič podporovat mechanizmus DMA bus mastering bez podpory scatter/gather I/O jako jediný způsob přenosu uložených paketů z interního bufferu řadiče do hlavní paměti počítače. Navrhněte a detailně popište HCI pro tuto síťovou kartu tak, aby si její ovladač mohl vždy vyžádat přenos minimálně jednoho celého paketu. Počítejte s tím, že pokud ovladač nestíhá „stahovat“ data paketů ze síťové karty, tak korektní chování řadiče je, že začne nejstarší pakety v interním bufferu přepisovat pakety nejnověji přijatými. Pro HCI máte k dispozici pouze I/O adresy z následujícího rozsahu `$1000` až `$2000`, a IRQ 10 až 12.

Otázka č. 2

Víme, že v proměnné x jsou všechny bity rovné 0, pouze bit 0 a bit 4 mají hodnotu 1. Zapište v šestnáctkové soustavě hodnotu proměnné x pro provedení níže uvedeného kódu v Pascalu:

```
x := x + (
($AABBCCDDEEFF0011 or
($0005500000000000 shl 8)) and (not(1) xor
65536));
```

Otázka č. 3

Víme, že jedna z hlavních funkcí vyššího programovacího jazyka je odstínění programátora od specifik architektury konkrétního cílového procesoru. Je tedy někdy potřeba při programování ve vyšším programovacím jazyce (např. v Pascalu) znát tzv. *endianitu* cílového procesoru, kde náš program poběží? Pokud ne, tak vysvětlete proč. Pokud ano, tak na příkladu vysvětlete proč.

Otázka č. 4

V kódování Unicode existuje následující přiřazení kódů jednotlivým znakům: kód 10Ch pro znak ‚Č‘, kód 61h pro znak ‚a‘, kód 73h pro znak ‚s‘.

Od adresy 0 chceme do paměti uložit *null-terminated* řetězec „Čas“ (bez uvozovek) v kódování UTF-16 ve variantě little endian. V šestnáctkové soustavě zapište hodnoty jednotlivých bytů paměti od adresy 0, které v sobě budou obsahovat část výše uvedeného stringu v daném kódování.

Otázka č. 5

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 51. 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. 6

Předpokládejte, že bychom chtěli pro OS určený pro 32-bitové CPU s podporou stránkování navrhnout mechanizmus sdílení fyzické paměti mezi procesy, které používají jednu a tu samou DLL knihovnu. Tento mechanizmus má být ale pro všechny procesy transparentní a nemá porušit oddělení jejich virtuálních adresových prostorů. Proto bychom potřebovali pro stránky, které obsahují kód nějaké DLL knihovny, zařídit, aby do nich žádný z procesů nemohl zapisovat (tedy explicitně chceme zakázat samomodifikující se kód – při pokusu o takovou operaci má dojít k ukončení daného vlákna). Dále bychom pro stránky, které obsahují globální data nějaké DLL knihovny, potřebovali naimplementovat tzv. mechanizmus *copy-on-write*: tedy dokud všechny procesy z nějaké stránky s globálními daty DLL knihovny data jen čtou, tak všechny procesy sdílí stejný rámec fyzické paměti. Když má v nějakém procesu dojít k zápisu do takovéto sdílené stránky, tak je potřeba pouze pro tento proces vytvořit její kopii do nového rámce (již nesdíleného s jinými procesy), a samotnou operaci zápisu do stránky povolit až do tohoto nového soukromého rámce.

Navrhněte jak tento mechanizmus v OS naimplementovat. Předpokládejte CPU, které má v každé položce stránkovací tabulky všechna typická data popisující mapování stránky. Dále předpokládejte, že v každé takové položce jsou ještě 4 B nevyužitého místa, které můžete použít pro své potřeby. V Pascalu (pseudokódu) pak zapište algoritmus, který bude třeba implementovat do obsluhy výpadku stránky (pro jednoduchost předpokládejte, že výpadky stránky vznikají jen v souvislosti s popsaným mechanizmem, a nikdy jindy).

Otázka č. 7

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*.

Otázka č. 8

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, A*n* = libovolný z registrů A0 až A7, D*n* = 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>*,A*n*	32	kopie do adr. registru
ADD	D*n*,*<op>* *<op>*,D*n*	8, 16, 32	přičtení datového nebo k datovému reg.
ADDA	*<op>*,A*n*	32	přičtení k adr. registru
SUB	D*n*,*<op>* *<op>*,D*n*	8, 16, 32	odečtení hodnoty
SUBA	*<op>*,A*n*	32	odečtení od adr. reg.
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
- D*n* operace s obsahem registru D*n*
- (A*n*) operace s pamětí daná obsahem registru A*n*
- *imm*(A*n*) operace s pamětí, cílová adresa je daná součtem obsahu registru A*n* a hodnoty *imm*
- -(A*n*) tzv. predekrementace: jako součást instrukce je hodnota v registru A*n* zmenšena o velikost prováděné operace v bytech, a nová hodnota v registru A*n* slouží jako cílová adresa
- (A*n*)+ tzv. postinkrementace: aktuální hodnota registru A*n* slouží jako cílová adresa, po provedení instrukce je ale hodnota v registru A*n* 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 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 `0xFFFF0002`, a v paměti od adresy `0x00100000` bude:
`00 00 00 07 00 00 00 00 00 00 00 03`

Úloha: Následující tělo hlavního programu v Pascalu přepište do assembleru Motorola 68000 s využitím tam běžné volací konvence (parametry na zásobníku uložené zprava doleva, návratová hodnota v D0) a s využitím jen popsaných instrukcí (velikost typu `Integer` je 16-bitů, proměnná `Vysledek` leží na adrese `0x00010000`):

```
var Vysledek : Integer;
function Suma(a, b, c : Integer) : Integer;
{ hlavní program }
begin
    Vysledek := 10;
    Vysledek := Vysledek + Suma(1, 2, 3);
end.
```

Otázka č. 9

Upravte kód následující procedury `Sort` tak, aby na dvou procesorovém systému mohla běžet netriviálně rychleji než na systému jednoprocesorovém:

```
type PLongint = ^Longint;
procedure Quicksort( { neužívá glob. prom. }
    pole : PLongint; n : Longint);
procedure Merge(     { neužívá glob. prom. }
    zdroj1 : PLongint; zdroj2 : PLongint;
    n1, n2 : Longint; cil : PLongint);
{ 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 procedur a funkcí pro podporu vícevláknového programování, které budete požadovat od OS. Ke každé napište stručný popis jejího očekávaného chování.

Otázka č. 10

Pod MS-DOSem jsme na počítači s CPU Intel 8088 kompatibilním (16-bit CPU, obecné registry AX, BX, CX, DX, 16 & 16-bit [`seg:ofs`] logické adresy, 20-bit fyzická adresa = `seg*16+ofs`, dva 16-bit registry `CS:IP` tvořící PC) jsme v řádkovém debuggeru vytvořili následující strojový kód nějaké funkce (argument i návratová hodnota v registru BX), kterou jsme si otestovali, že funguje správně:

```
-u 5000:0
5000:0000 B80000          MOV     AX,0000
5000:0003 39C3            CMP     BX,AX
5000:0005 7403            JZ      000A
5000:0007 BB0500          MOV     BX,0005
5000:000A 83C302          ADD     BX,+02
5000:000D C3              RET
```

Tento strojový kód uložíme jako binární obraz paměti od adresy `5000:0000` do 14 bytů dlouhého souboru. Pokud po restartu počítače nahrajeme tento obraz na adresu `5000:01FF` bude uvedená funkce stále stejně a bez chyb fungovat? Vysvětlete proč!