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 a do jejich záhlaví napsat i/N (kde i je číslo listu, N je celkový počet odevzdaných listů).

Otázka č. 1
(otázka za celkem 2 body)

Předpokládejte počítač s 32-bitovým paměťovým adresovým prostorem. V systému je nainstalována 256 kB velká paměť ROM, která je souvisle namapována na nejvyšší možné adresy v paměťovém adresovém prostoru počítače. Dále je v systému nainstalováno 0,5 GB paměti RAM, která je souvisle namapována od adresy `0` v paměťovém adresovém prostoru počítače, s výjimkou adres `170h` až `177h` na kterých jsou namapovány porty řadiče pevných disků pomocí mechanizmu MM I/O. Předpokládejte, že v Pascalu (případně v jazyce C) implementujete část firmware počítače, který bude uložený ve zmíněné paměti ROM. Napište implementaci funkce `Write` (a případně dalších procedur a funkcí, které budete potřebovat) s následujícím prototypem (viz níže), která má zařídit zapsání 1 sektoru dat (vždy `512` bytů) na pevný disk – parametr `sector` určuje číslo sektoru na disku, který se má zapsat; parametr `data` obsahuje ukazatel na `512` bytů dat v paměti, které se mají zapsat do daného sektoru (ukazatel samozřejmě obsahuje adresu 1. z 512 bytů). Funkce se vrátí ihned, jak to bude možné a nečeká na kompletní dokončení operace zápisu (tj. je asynchronní operací zápisu na pevný disk). Funkce vrací: (a) `True`, pokud došlo k úspěšnému spuštění operace zápisu, (b) `False`, pokud ještě probíhá předchozí operace zápisu na pevný disk a tedy požadovaný zápis není možné provést:

```
type
      PByte = ^Byte;
function Write(sector : Word;
               data : PByte) : Boolean;
procedure InitHarddisk;
procedure SetInterruptVector(
                intVec : Integer;
                handlerRoutine : Pointer);
```

Při inicializaci firmware počítače se mimo jiné volá i vaše procedura `InitHarddisk` (viz výše), do které můžete naimplementovat libovolnou inicializaci (např. vašich globálních proměnných), kterou budete potřebovat. Dále je vám k dispozici předpřipravená procedura `SetInterruptVector`, která do vektoru přerušení s číslem `intVec` nastaví adresu obslužné procedury předané v parametru `handlerRoutine`. Můžete očekávat, že od začátku do konce běhu obslužné procedury jsou zakázána všechna přerušení.

S řadičem pevného disku se komunikuje pomocí mechanizmu PIO. Pro iniciaci operace zápisu na pevný disk je třeba provést následující posloupnost zápisů do portů jeho řadiče (vždy je uvedena adresa, na které je daný port namapovaný):

1) 173h: horních 8 bitů čísla sektoru
2) 174h: spodních 8 bitů čísla sektoru
3) `177h`: 8-bitový identifikátor příkazu: příkaz `WRITE SECTOR` = hodnota `30h`

Poté je třeba vyčkat, až bude řadič připraven na přijímání dat – to řadič indikuje nastavením 7. bitu (číslované od 0), tzv. BSY (Busy), ve stavovém portu na adrese `177h` na hodnotu `0`. Poté je možné zapisovat jednotlivé byty (které mají být zapsány do vybraného sektoru) do datového portu na adrese `170h`. Zápis každého bytu do datového portu způsobí nastavení bitu BSY na hodnotu `1`. Před zápisem každého dalšího datového bytu je tedy třeba vždy vyčkat, až řadič opět oznámí svoji připravenost nastavením BSY na `0`.

Každé nastavení bitu BSY na hodnotu `0` je řadičem disku indikováno vyvoláním přerušení číslo `15`. V obsluze přerušení `15` je třeba ověřit, že přerušení opravdu vyvolal řadič disků a ne jiný zdroj, tj. že hodnota bitu BSY je opravdu `0`. Pokud přerušení pochází z jiného zdroje, je možné ho ignorovat.

Předpokládejte, že není zapnutá segmentace, ani stránkování. Předpokládejte, že typ `Word` slouží pro ukládání bezznaménkových celých čísel a jeho velikost je 16-bitů.

Otázka č. 2

Následující hodnotu:

`5,5625`

zapište jako 32-bitové reálné číslo s pohyblivou desetinnou čárkou. Mantisa je normalizována se skrytou 1 a zabírá spodních 23 bitů, pak následuje 8-bitový exponent uložený ve formátu s posunem (bias) +127 a 1 znaménkový bit. Výsledek zapište jako hodnoty jednotlivých bitů.

Otázka č. 3

Předpokládejme, že implementujeme operační systém poskytující podporu pro vícevláknové zpracování, dynamicky linkované knihovny a stránkování (tj. běží jen na procesorových platformách s podporou stránkování). Předpokládejme situaci, že více procesů běžících v takovém systému často používá stejnou dynamicky linkovanou knihovnu. Abychom ušetřili paměť, tak bychom chtěli, aby v takovém případě byl minimálně kód knihovny uložený v paměti RAM jen jednou. Je možné to nějak zařídit? Pokud ano, popište přesně, jakým způsobem by to bylo možné, a jak bude v takovém systému probíhat spouštění aplikace používající takovou „sdílenou" DLL. Pokud ne, popište přesně všechny důvody, proč není možné takový systém naimplementovat.

Otázka č. 4

Vysvětlete rozdíl mezi Harvardskou a von Neumannovou architekturou a popište výhody a nevýhody každé z nich. Pro každou z nich uveďte, zda se ještě používá v dnešních počítačích, a pokud ano, tak uveďte nějaký příklad.

Otázka č. 5

Procesor MOC 6502 je 8-bitový procesor se 16-bitovou adresovou sběrnicí (paměťovým adresovým prostorem) a s akumulátorovou architekturou. Registr akumulátoru má v assembleru jméno A. V příznakovém registru je příznak Carry (přenos) označený písmenem C.
Procesor 6502 má následující instrukce:

- LDA *`$adresa/#konstanta`* (Load Accumulator) – načtení jednobytové hodnoty na zadané adrese (argument instrukce uvozený $) *nebo* přímo zadané jednobytové konstanty (argument instrukce uvozený #) do akumulátoru.
- STA *`$adresa`* (Store Accumulator) – uložení hodnoty akumulátoru na zadanou adresu (argument instrukce)
- ADC *`$adresa/#konstanta`* (Add with Carry) – sečtení dvou 8-bitových čísel a příznaku C (druhý operand operace sčítání je argumentem instrukce)
- SBC *`$adresa/#konstanta`* (Subtract with Carry) – odečtení dvou 8-bitových čísel a negace příznaku C (druhý operand operace odečítání je argumentem instrukce)
- CLC (Clear Carry) – nastavení příznaku C na 0 (instrukce bez explicitních argumentů)
- SEC (Set Carry) – nastavení příznaku C na 1 (instrukce bez explicitních argumentů)

Přepište následující příkazy v Pascalu do ekvivalentní posloupnosti instrukcí strojového kódu procesoru 6502 (jednobytové proměnné A, B, C v sobě obsahují celá bezznaménková čísla a jsou uloženy na následujících adresách: A = A000h, B = B000h, C = C000h). Všechny operace chceme provést v celých číslech s 8-bitovou přesností bez znaménka:

```
B := 15 + B;
A := C – B – A;
```

Otázka č. 6

Předpokládejte, že máte procesor, který má v sobě zabudovanou 1 MB velkou cache, a na kterém poběží váš program napsaný v Pascalu. Je potřeba při programování v Pascalu někdy znát velikost procesorové cache, nebo je její velikost pro běžné programování irelevantní (tj. její velikost je třeba znát např. jen při programování přímo v assembleru, resp. strojovém kódu)? Vysvětlete proč!

Otázka č. 7

Předpokládejte programové prostředí, které poskytuje podporu pro vícevláknové zpracování. Předpokládejte, že hlavní procedura nějakého vlákna doběhne do konce (tj. toto vlákno je ve stavu těsně před provedením instrukce RET [instrukce návratu z podprogramu] na konci jeho hlavní procedury). Popište a vysvětlete, co se bude dít potom (jaký kód bude procesor dále zpracovávat a co tento kód bude dělat).

Otázka č. 8

Vysvětlete, co je memory model, a v jakých situacích je jeho znalost důležitá.

Otázka č. 9

Následující program zapsaný v jazyce Pascal můžete pomocí překladače Free Pascal přeložit a spustit jak na OS Linux na platformě ARM, tak i na OS Windows 7 na platformě x86. Popište a vysvětlete, z jakého důvodu je to možné a jaké všechny kroky je třeba provést, abyste z původního zdrojového souboru získali spustitelný soubor. Do výkladu zahrňte zdůvodnění, zda a proč pro daný scénář bude potřeba více různých spustitelných souborů daného programu, nebo zda a proč bude dostačovat pouze jeden.

```
program Faktorial;
var
  f, n : integer;

begin
  ReadLn(n);
  f := 1;
  while n > 1 do begin
    f := f * n;
    Dec(n);
  end;
  WriteLn('Faktorial je ', f);
end.
```