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

Vysvětlete, co to je tzv. *třístavová logika* (*tri-state logic*). Dále nakreslete příklad elektrického zapojení sériové linky s jedním vysílajícím a s jedním přijímajícím. Předpokládejte, že vysílající strana může vysílat pouze hodnotu `0` nebo `1`, ale na přijímající straně bychom chtěli umět detekovat nejen hodnotu vysílanou z druhé strany, ale také odpojení vysílače od sériové linky. Dále předpokládejte, že přijímací strana poskytuje napájení i straně vysílající.

Otázka č. 2

V typické instalaci operačního systému Windows 7 se v kořenovém adresáři systémového disku (ze kterého OS nabootoval) nachází soubor `pagefile.sys`. Vysvětlete, co je nejspíš obsahem takového souboru, a proč se při běhu počítače někdy jeho velikost zvětšuje. Dokážeme nějak přibližně odhadnout, na čem bude záviset jeho velikost? Bude s větší velikostí operační paměti počítače velikost tohoto souboru spíše větší nebo naopak menší?

Společná část pro otázky označené X

Procesor 6502 je **8-bitový little endian** mikroprocesor s akumulátorovou architekturou a s **16-bitovým paměťovým adresovým prostorem**. Kromě registru akumulátoru (v assembleru značený A) má procesor ještě dva pomocné registry X a Y. Procesor má HW podporu pro implementaci volacího zásobníku – volací zásobník musí ležet na adresách mezi `$100` a `$1FF`. Zásobník roste dolů, offset **prvního volného místa** na zásobníku je uložen v **8-bitovém** registru S (pozor: ve skutečnosti první volné místo na zásobníku leží na adrese `$100 + S`). Procesor 6502 má následující instrukční sadu (∀ instrukce má 1 byte opcode): **LDA** ***arg*** (Load Accumulator), **STA** ***arg*** (Store Accumulator), **PHA** (Push Accumulator) , **PLA** (Pull Accumulator = ekvivalent operace POP běžné na jiných procesorech), **NOP** (No Operation), **SEC** (Set Carry), **CLC** (Clear Carry), **ADC** ***arg*** (Add with Carry), **SBC** ***arg*** (Subtract from A with Borrow), **INX** (INrement X – zvětší obsah registru X o jedna), **DEC** ***addr*** (DECrement – zmenší 8-bit cílovou paměťovou lokaci argumentu o jedna), **JSR** ***addr*** (Jump to SubRoutine – ekvivalent u jiných procesorů běžné instrukce CALL), **RTS** (Return from SubRoutine – ekvivalent RET), **JMP** ***addr*** (JuMP – nepodmíněný skok), **BEQ** ***addr*** (Branch if EQual), a 6 instrukcí pro kopírování hodnoty (Transfer) mezi registry (2. písmeno jména instrukce = zdrojový registr, 3. písmeno = cílový registr): **TXA**, **TAX**, **TYA**, **TAY**, **TSX**, **TXS**. Argument instrukcí (pokud ho mají) může být jedna z následujících variant: ***#číslo*** (= hodnota immediate), ***číslo*** (= adresa v paměti), ***číslo*****,X** (= přímá adresa v paměti získaná výpočtem – výsledná adresa v paměti se získá jako *číslo* + X, tj. výsledek součtu hodnoty *číslo* a hodnoty uložené v registru X), podobně ***číslo*****,Y** (= přímá adresa v paměti, tj. adresa *číslo* + Y).

Předpokládejte, že od adresy `$609` je v paměti počítače Commodore C64 (má procesor 6502) uložena funkce `Foo`, jejíž strojový kód byl získán překladem z Pascalu pomocí vhodného překladače – její deklarace v Pascalu byla následující:

```
function Foo(a : byte; b : byte) : byte;
```

Dále je od adresy `$63c` uložena funkce `Bar`, jejíž deklarace v Pascalu byla následující:

```
function Bar(n : byte; a : byte) : byte;
```

Obraz paměti našeho počítače Commodore C64 mezi adresami `$609` až `$065b` je následující (zobrazen hex dump včetně výsledku disassembleru do assembleru 6502):

```
$0609 ba       TSX
$060a 8a       TXA
$060b a8       TAY
$060c ad 5b 06 LDA $065b
$060f 48       PHA
$0610 a9 03    LDA #$03
$0612 48       PHA
$0613 b9 03 01 LDA $0103,Y
$0616 48       PHA
$0617 20 3c 06 JSR $063c
$061a ba       TSX
$061b e8       INX
$061c e8       INX
$061d 9a       TXS
$061e 18       CLC
$061f 79 00 01 ADC $0100,Y
$0622 99 00 01 STA $0100,Y
$0625 a9 05    LDA #$05
$0627 48       PHA
$0628 b9 04 01 LDA $0104,Y
$062b 48       PHA
$062c 20 3c 06 JSR $063c
$062f ba       TSX
$0630 e8       INX
$0631 e8       INX
$0632 9a       TXS
$0633 18       CLC
$0634 79 00 01 ADC $0100,Y
$0637 99 00 01 STA $0100,Y
$063a 68       PLA
$063b 60       RTS
```

```
$063c ba       TSX
$063d bd 04 01 LDA $0104,X
$0640 48       PHA
$0641 bd 03 01 LDA $0103,X
$0644 38       SEC
$0645 e9 01    SBC #$01
$0647 f0 10    BEQ $0659
$0649 bd 00 01 LDA $0100,X
$064c 18       CLC
$064d 7d 04 01 ADC $0104,X
$0650 9d 00 01 STA $0100,X
$0653 de 03 01 DEC $0103,X
$0656 4c 41 06 JMP $0641
$0659 68       PLA
$065a 60       RTS
$065b 04       ???
```

Otázka č. 3 (X)

Napište v Pascalu bez použití inline assembleru kód fce (i s deklarací), která by mohla být běžným překladačem přeložena do kódu fce `Bar` uvedené výše v assembleru 6502 (ta od adresy `$63c`).

Otázka č. 4 (X)

Vysvětlete, co a proč je obecně součástí definice volací konvence nějaké funkce. Kdo a kdy volací konvenci definuje? Popište, jaká je asi volací konvence funkce `Bar` ve výše uvedeném příkladu.

Otázka č. 5 (X)

Předpokládejte, že bychom změnili deklaraci funkce `Bar` a její nová deklarace by vypadala následujícím způsobem:

```
function Bar(n : byte; a : shortint) : byte;
```

kde typ `shortint` je v Pascalu znaménkový 8-bitový celočíselný typ. Změnil by se při použití běžného překladače Pascalu strojový kód vygenerovaný z funkce `Bar` oproti variantě s bezznaménkovým parametrem `a`, pokud bylo tělo funkce v Pascalu v obou variantách „na chlup“ stejné? Vysvětlete proč.

Otázka č. 6 (X)

Napište v Pascalu program jednoduché virtual machine jako interpret simulující chování procesoru 6502. Do simulace zahrňte registry PC, A, X, S, SR (Status Register = ekvivalent příznakového registru běžného na jiných procesorech), a simulaci instrukcí TAX (opcode $AA), TXA, PHA, kde instrukce TAX i TXA mění obsah příznakového registru dle výsledku provedené operace, instrukce PHA příznakový registr nemění. Předpokládejte, že bychom v budoucnu do programu dopsali simulaci i všech ostatních instrukcí.
Program bere na příkazové řádce dva parametry:
1. parametr – dostupný jako stringový výsledek běžné Pascal funkce `ParamStr(1)` – je jméno souboru s obrazem části paměti, který obsahuje strojový kód, který má naše virtual machine provádět.
2. parametr – dostupný jako stringový výsledek běžné Pascal funkce `ParamStr(2)` – je celé číslo v rozsahu 0 až 65535 zapsané v desítkové soustavě, které reprezentuje adresu v simulované paměti, od které dál se má nahrát strojový kód ze souboru předaného jako první parametr.
Program má po simulovaném vykonání každé instrukce vypsat na standardní výstup jméno právě provedené instrukce a obsah všech 5 výše uvedených registrů.

Otázka č. 7 (X)

Předpokládejte, že deklarace funkce Bar zůstala:

```
function Bar(n : byte; a : byte) : byte;
```

ale deklaraci funkce Foo jsme změnili na:

```
function Foo(a : byte; b : byte) : word;
```

a zároveň globální proměnná `p1` od adresy `$65f` je nyní 16-bitová (= původní 8-bitová proměnná z adresy `$65b`). Pokud bychom chtěli, aby funkce `Foo` nyní prováděla 16-bitové sčítání 16-bitové proměnné `p1` s 8-bitovými výsledky volání funkce `Bar`, jak by bylo třeba kód funkce `Foo` v assembleru opravit? Začátek funkce `Foo` jsme již opravili, nyní nás ale zajímá náhrada instrukcí na adresách `$622` až `$628` (nevadí, pokud váš nový kód bude mít jinou délku než kód původní, tj. že bude zabírat méně nebo více místa než stávající kód do adresy `$628`). Nový kód napište v assembleru (samozřejmě k němu nemusíte doplňovat strojový kód).

```
$0609 ba       TSX
$060a 8a       TXA
$060b a8       TAY
$060c ad 60 06 LDA $0660     ; již opraveno
$060f 48       PHA
$0610 ad 5f 06 LDA $065f
$0613 48       PHA
$0614 a9 03    LDA #$03
$0616 48       PHA
$0617 b9 03 01 LDA $0103,Y
$061a 48       PHA
$061b 20 40 06 JSR $0640
$061e ba       TSX
$061f e8       INX
$0620 e8       INX
$0621 9a       TXS
$0622 18       CLC           ; je třeba opravit
$0623 79 00 01 ADC $0100,Y
$0626 99 00 01 STA $0100,Y
...
$065f 04       ???
$0660 00       BRK
```

Společná část pro otázky označené Y

Předpokládejte, že navrhujeme novou řadu počítačů založených na 8-bitovém procesoru 6502, kde procesor 6502 podporuje pouze paměťový adresový prostor o velikosti 16-bitů. Na základní desce počítače chceme mít kromě paměti EEPROM s uloženým firmwarem počítače připojeno také celkem 256 KiB operační paměti typu DRAM.

Otázka č. 8 (Y)

Navrhněte jakým způsobem principiálně takové množství paměti DRAM do počítače zapojit. Pokud k tomu bude třeba nějakých řadičů, tak v této situaci popište jejich principiální zapojení a jejich chování. Pokud takové řadiče budou ovladatelné ze software počítače, tak také navrhněte a popište vhodné HCI takových řadičů (navrhněte veškeré potřebné paměťově mapované registry a vymyslete ke každému nějakou možnou adresu). Nevadí nám, pokud část paměti DRAM bude shadowovaná pamětí EEPROM s firmwarem, nebo registry řadičů, a tedy bude nepřístupná.

Otázka č. 9 (Y)

Předpokládejte, že pro uvedený počítač programujeme v Pascalu část jeho firmware – navíc předpokládejte knihovna RTL našeho překladače umí přímo pracovat se standardními řadiči, které v počítači budeme mít (tj. např. bude správně implementovat proceduru `WriteLn` tak, aby text vypisovala přímo do framebufferu grafické karty) – ale pozor: RTL přímo nepodporuje práci s žádným s řadičů, které jste případně navrhli v otázce 8. Pro zapojení z otázky 8 nyní napište v Pascalu program, který na standardní výstup vypíše v desítkové soustavě aktuální obsah každého bytu z připojených 256 KiB paměti DRAM (obsah shadowovaných paměťových buněk DRAM nemusí program správně vypisovat).

Otázka č. 10

Předpokládejte, že nějakým běžným překladačem Pascalu překládáme program (s níže uvedeným typem záznamu a globální proměnnou `x`) pro 32-bitový **big-endian** procesor Motorola 68000. Použitý překladač Pascalu je navržený s vědomím, že všech procesory dané řady jsou pouze 32-bitové:

```
type
  TZaznam = record
    a : byte;
    b : word;       { 16-bit celé bezznam. číslo }
    c : byte;
    d : longint;    { 32-bit celé znaménkové číslo }
    e : byte;
    f : string[16]; { char: znak 8-bit rozš. ASCII }
  end;
var
  x : TZaznam;
begin
  ...
end.
```

Pokud je proměnná `x` po nahrání programu do paměti počítače uložená od adresy `0x00001F00`, tak detailně popište, jaké všechny adresy v paměti zabírají jednotlivé části proměnné `x`. Pro každou takovou adresu napište, kterou částí záznamu je použitá. Vaše řešení vysvětlete a zdůvodněte – proč by ho asi překladač Pascalu použil?