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).

Společná část pro otázky označené X

Předpokládejte, že máme fiktivní 8-bitový procesor s 16-bitovým fyzickým i virtuálním adresovým prostorem a s podporou pro stránkování. Fyzický adresový prostor je rozdělen na 16 rámců po 4 kB, virtuální adresový prostor je rozdělen na 16 stránek také po 4 kB. Procesor používá jednoúrovňové stránkovací tabulky, stránkovací tabulka může ležet na libovolné adrese a její bázová adresa se nastavuje do registru PTBR. Jeden záznam tabulky stránek má velikost 1 byte a má následující formát:

7:4	3	2	1	0
Frame Number	D	U/S#	R/W#	P

kde jednotlivé bity mají následující význam:

- P = present (pokud je 0, jsou bity 1 až 7 procesorem ignorovány [OS si v takové situaci do nich může uložit libovolné pomocné informace, a žádný z dalších bitů 1 až 7 pak nemá původní význam]),
- R/W# = Read-only/Read-write,
- U/S# = User/Supervisor,
- D = Dirty, nastavte na 0.

Tabulka vektorů přerušení (IVT) má u tohoto procesoru 256 položek, obsluha libovolného z těchto 256 přerušení může ležet na libovolné adrese 16-bitového adresového prostoru. Bázová adresa IVT může být libovolná zarovnaná na velikost jednoho záznamu IVT. Bázová adresa IVT se nastavuje do registru IVTR procesoru.

Otázka č. 1 (X)

Na počítači s výše uvedeným procesorem máme nabootovaný jednoduchý operační systém s podporou pro běh více procesů – OS podporuje stránkování a odděluje od sebe jednotlivé procesy mezi sebou a i od jádra OS tak, jak je běžné v moderních OS jako Windows nebo Linux. Jádro OS má monolitickou architekturu, kdy všechny ovladače jsou již součástí samotného obrazu jádra. Kód celého jádra OS v obrazu na disku zabírá 9192 bytů, globální proměnné a konstanty jádra zabírají 3808 bytů. Všechny dynamické datové struktury jádra jsou alokovány na jeho vlastní haldě. Předpokládejte situaci, kdy je aktuální velikost zaalokovaného místa na této systémové haldě 4040 bytů. Předpokládejte, že je v systému spuštěn jeden jediný proces, který reprezentuje běžící aplikaci s 6100 byty kódu, 516 byty globálních proměnných a konstant, a haldou, která celkem zabírá 3000 bytů. V procesu běží jediné vlákno s aktuálně spotřebovanými 100 byty zásobníku. Rozmyslete, jak by v této situaci mohl vypadat obsah stránkovací tabulky tohoto procesu (navrhněte libovolné vhodné rozložení stránek), a napište v šestnáctkové soustavě obsah každého z 16 bytů této stránkovací tabulky. U každého záznamu označte, jaká data budou ve stránce uložena.

Otázka č. 2 (X)

Pokud OS stránkovací soubor využívá nejjednodušším možným způsobem, tak jaká je na výše uvedeném systému jeho maximální velikost? Vysvětlete proč.

Otázka č. 3 (X)

Předpokládejte, že proces z otázky 1 zavolá API funkci jádra OS za situace, kterou jste vyznačili ve své odpovědi na otázku 1. Jako součást provedení této API funkce potřebuje kernel zapsat hodnotu $FF do paměťově mapovaného registru řadiče klávesnice na adrese $0100. Pokud bychom takové jádro OS programovali v Pascalu, jak bychom v Pascalu takový zápis do registru zařízení zapsali? Náš OS nemá HAL! Bude potřeba před samotným zápisem provést ještě nějakou speciální akci (např. úpravu nějakých datových struktur jádra)? Vysvětlete.

Otázka č. 4

Co nejefektivněji rozhodněte, která z následujících čísel jsou dělitelná 32. Stručně napište, jak jste postupovali.

(a) 0x5b20897c4180a8c0 (b) 0xf665b8981844271d
(c) 0x2004eb774a522c5c (d) 0x0ae4d8bfee61b892
(e) 0x064abd7235adc032 (f) 0x6d85f8df77553070
(g) 0x8a99a2aeaaed49dc (h) 0x00000030acf4dca7
(j) 0x071e4bfa79dab636 (k) 0x3a84001852f10000

Otázka č. 5

Předpokládejte, že máme N-bitové celé nezáporné číslo A, a M-bitové celé nezáporné číslo B, kde N < 16 , M < 16. Víme, že číslo A je uložené v N spodních bitech proměnné iA, a číslo B je uložené v M spodních bitech proměnné iB. Obsah ostatních bitů proměnných iA a iB je *neznámý*. Dále máme v programu proměnné N a M, které určují rozsah A a B. Vaším úkolem je v Pascalu napsat kód níže uvedené funkce, který nastaví obsah proměnné oC tak, že v nejnižších N bitech je uložena hodnota A, v následujích M bitech (od N. bitu) je uložena hodnota B, zbývající (nejvyšší) bity proměnné oC mají být nulové. Všechny proměnné iA, iB, N, M, oC jsou 32-bitové.

```
function Join(
  iA : longword; N : longword;
  iB : longword; M : longword;
) : longword;
var
  oC : longword;
begin
  { sem doplňte váš kód }
  Join := oC;
end;
```

Otázka č. 6

Vysvětlete, zda je typická paměť RAM „*random access*“, a zda je typická paměť ROM „*read-only*“? Jaký je hlavní rozdíl mezi dnes běžnou pamětí typu RAM a běžnou pamětí typu ROM? Vysvětlete.

Otázka č. 7

Jak procesor běžného desktopového počítače ví, jaký kód má po zapnutí napájení počítače provádět? Kde takový kód vezme? Dále vysvětlete, co to je *boot loader*, a co rámcově provádí jeho kód. Kde CPU tento kód vezme, a jak ví, že ho má provádět?

Společná část pro otázky označené Y

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 `$A00` je v paměti počítače Apple II (má procesor 6502) uložena funkce `Foo` s neznámým počtem parametrů (parametry se předávají na zásobníku zleva doprava) a návratovou hodnotou stejného typu (předává se v registru A). Obraz paměti našeho počítače Apple II mezi adresami `$A00` až `$A2A` je následující (zobrazen hex dump včetně výsledku disassembleru do assembleru 6502):

```
$0A00 ba       TSX
$0A01 a9 00    LDA #$00
$0A03 48       PHA
$0A04 a9 01    LDA #$01
$0A06 48       PHA
$0A07 bd 03 01 LDA $0103,X
$0A0A 38       SEC
$0A0B fd ff 00 SBC $00FF,X
$0A0E f0 16    BEQ $0A26
$0A10 bd 00 01 LDA $0100,X
$0A13 18       CLC
$0A14 7d ff 00 ADC $00FF,X
$0A17 9d 00 01 STA $0100,X
$0A1A bd ff 00 LDA $00FF,X
$0A1D 18       CLC
$0A1E 69 01    ADC #$01
$0A20 9d ff 00 STA $00FF,X
$0A23 4c 07 0A JMP $0A07
$0A26 bd 00 01 LDA $0100,X
$0A29 9a       TXS
$0A2A 60       RTS
```

Otázka č. 8 (Y)

Napište v Pascalu bez použití inline assembleru kód fce (i s deklarací), která by mohla být běžným překladačem přeložena do kódu fce `Foo` uvedené výše v assembleru 6502 (ta od adresy `$A00`).

Otázka č. 9 (Y)

Napište v Pascalu (případně v C/C++ nebo v C#) 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: `CLC`, `LDA` ve variantě s immediate operandem (tj. instrukce `LDA #$??`), a `LDA` ve variantě s operandem s přímou adresou získanou součtem s registrem `X` (tj. instrukce `LDA $????,X`), kde tyto obě vámi implementované varianty instrukce `LDA` mění obsah příznakového registru dle výsledku provedené operace. 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. 10 (Y)

Předpokládejme, že máme speciální variantu procesoru 6502 s jednoduchou paralelní systémovou sběrnicí, která má sdílené adresové a datové vodiče. Na systémové sběrnici může být pouze jediný master, a to samotné CPU. Dále předpokládejte, že procesor právě dokončil samotné zpracování instrukce `ADC $00FF,X` na adrese `$0A14`, a právě zvětšuje obsah registru `PC` o `3` jako poslední krok jejího zpracování. Předpokládejte, že po dokončení zpracování instrukce `ADC` je v registru `A` hodnota `$E7`, a v registru `X` hodnota `$F0`. Za předpokladu, že procesor 6502 nemá žádnou cache, ani prefetch buffer, tak nakreslete časový diagram všech přenosů po systémové sběrnici od uvedeného okamžiku až do doby, kdy dojde ke kompletnímu dokončení následující instrukce `STA $0100,X` a dalšímu zvětšení registru `PC` opět o `3`. Nakreslete stav všech vodičů důležitých pro přenos. Vodiče, u kterých to dává smysl, můžete v obrázku seskupit do jednoho řádku diagramu.