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

Naimplementujte v Pascalu funkci `GetCharCount` s následujícím prototypem:

```
type
  PUtf8 = ^byte;
function GetCharCount(text : PUtf8) : word;
```

která jako parametr `text` bere ukazatel na null-terminated řetězec v kódování UTF-8, a vrátí počet Unicode znaků (každý běžný, řídící, i combining znak se do výsledného počtu započítává, tedy se **nepočítají** celé grafémy, ale opravdu jednotlivé znaky), ze kterých se tento řetězec skládá. Pokud je znak v UTF-8 kódován delší než 1 bytovou sekvencí, tak počet bytů jednoho znaku zjistíte z 1. bytu takové sekvence: pro každý N-bytový znak (kde N > 1) platí, že nejvyšších N bitů 1. bytu je rovno `1`, pak následuje jeden bit rovný `0`. Druhý až N-tý byte sekvence vypadají tak, že bit `7` je roven `1`, bit `6` je roven `0`.

Společná část pro otázky označené X

Předpokládejte, že máme čistě naformátovaný souborový systém FAT12 používající tabulku FAT – kde, jeden záznam ve FAT má 12 bitů, hodnota `0` reprezentuje volný cluster, konec souboru reprezentuje maximální hodnota. Souborový systém je na disketě se `128 B` sektory, velikost jednoho clusteru jsou 4 sektory. První cluster použitelný pro data souborů (tj. 1. datový cluster) je označený číslem `1`, a odpovídá mu první záznam ve FAT. Pro obsah kořenového adresáře jsou napevno vyhrazeny 4 clustery před 1. datovým clusterem – **pozor:** kořenový adresář je u FAT12 jediný soubor, který má pevný začátek, a veškeré jeho clustery jsou předalokovány kontinuálně za sebou a **nejsou** spravovány FAT tabulkou. Pro každý adresář platí, že jeden záznam adresářové položky má následující formát:

Offset	|B|	Popis (všechny hodnoty jsou LE byte order)
`0x00`	8	ASCII řetězec se jménem souboru (zprava dorovnaný mezerami = ASCII kód `$20`). Pokud je `0`. byte roven `0x00` nebo `0xE5`, tak je tento adresářový záznam nepoužitý a byty 1-31 nenesou platné informace.
`0x08`	3	ASCII řetězec přípony souboru (zprava dorovnaný mezerami). Soubory bez přípony mají tuto položku vyplněnou 3 mezerami.
`0x0B`	1	Sada příznaků: bit 4: `0` = soubor, `1` = adresář
`0x0C`	10	Rezervováno (má být `0`)
`0x16`	2	Čas poslední modifikace souboru: bity 15-11: hodina (0-23) bity 10-5: minuta (0-59) bity 4-0: sekund / 2 (0-29)
`0x18`	2	Datum poslední modifikace souboru: bity 15-9: rok (0=1980) bity 8-5: měsíc (1-12) bity 4-0: den (1-31)
`0x1A`	2	Číslo prvního datového clusteru souboru
`0x1C`	4	Délka souboru v bytech

Otázka č. 2 (X)

Nakreslete konečný obsah prvních 16 záznamů FAT tabulky po provedení následujících operací (předpokládejte, že pokud je potřeba další volný cluster pro data souboru, tak se v souborovém systému vybere první volný datový cluster s nejnižším číslem; operace „zápis `N` bytů do `X`“ se chápe jako připsání `N` bytů za poslední byte souboru `X`):

1) Vytvoření prázdného souboru `/A.TXT`
2) Vytvoření prázdného adresáře `/DIR`
3) Vytvoření prázdného souboru `/DIR/B.TXT`
4) Zápis `2 KiB` do `/A.TXT`
5) Zápis `128 B` do `/DIR/B.TXT`
6) Vytvoření prázdného souboru `/DIR/C.TXT`
7) Zápis `1 KiB` do `/DIR/C.TXT`
8) Zápis `1 B` do `/A.TXT`
9) Smazání celého souboru `/DIR/C.TXT`
10) Zápis `512 B` do `/DIR/B.TXT`

Otázka č. 3 (X)

Předpokládejte, že obsah souborového systému odpovídá stavu pro provedení všech operací z otázky 2, a že všechny soubory a adresáře byly naposledy modifikovány dnes (viz záhlaví zadání) a teď (viz hodiny na zdi posluchárny – tento čas i zapište do písemky ve formátu `hh:mm:ss`). Zapište v šestnáctkové soustavě obsah prvních `32` bytů kořenového adresáře. Znak `A` má v ASCII kódování kód `$41`.

Otázka č. 4

Předpokládejte, že chceme navrhnout část řadiče 32-bit PCI 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 `8192 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 jako jediný způsob přenosu uložených paketů z interního bufferu řadiče do hlavní paměti počítače.

Navrhněte a detailně popište HCI (Host Controller Interface) 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 `$10FF`, a jen IRQ 11.

Otázka č. 5

Předpokládejte nějaký segment běžné PCI sběrnice, na které jsou připojena 3 zařízení chovající se jako *master*. Na příkladu časového diagramu vysvětlete, jakým způsobem zde bude probíhat arbitrace sběrnice, a kdo a jak jí bude provádět. Jaká je výhoda oproti arbitraci běžné na I^2C?

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), **NOP** (No Operation), **SEC** (Set Carry), **CLC** (Clear Carry), **ADC** ***arg*** (Add with Carry), **SBC** ***arg*** (Subtract from A with Borrow), **ASL A** (Shift Left – provede posun hodnoty v A o jedna doleva), **JSR** ***addr*** (Jump to SubRoutine – ekvivalent u jiných procesorů běžné instrukce CALL), **RTS** (Return from SubRoutine – ekvivalent RET), **BNE** ***addr*** (Branch if Not 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 `$514` je v paměti počítače Apple II (má procesor 6502) uložena funkce se dvěma parametry (parametry se předávají na zásobníku zleva doprava) a návratovou hodnotou stejného typu (předává se v registru A) – viz kód níže vpravo; dále je od adresy `$537` v paměti uložen kód, který je ukázkou volání uvedené funkce s parametry 5 a 3 – viz kód níže vlevo (instrukce NOP na konci reprezentují místo pro budoucí kód, který bude zpracovávat hodnotu spočítanou voláním uvedené funkce):

```
$0537 LDA #$05
$0539 PHA
$053a LDA #$03
$053c PHA
$053d JSR $0514
$0540 TAY
$0541 TSX
$0542 TXA
$0543 CLC
$0544 ADC #$02
$0546 TAX
$0547 TXS
$0548 TYA
$0549 NOP
$054a NOP
$054b NOP
```

```
$0514 TSX
$0515 LDA $0103,X
$0518 BNE $051e
$051a LDA $0104,X
$051d RTS
$051e LDA $0104,X
$0521 PHA
$0522 LDA $0103,X
$0525 SEC
$0526 SBC #$01
$0528 PHA
$0529 JSR $0514
$052c TAY
$052d TSX
$052e TXA
$052f CLC
$0530 ADC #$02
$0532 TAX
$0533 TXS
$0534 TYA
$0535 ASL A
$0536 RTS
```

Otázka č. 6 (Y)

Předpokládejte, že v registru S je hodnota `$FF` (tedy „prázdný“ zásobník) a byty na adresách `$100` až `$1FF` jsou vyplněny `0`. Pokud bychom v tomto stavu provedli skok na adresu `$537`, jaký by byl obsah paměti na adresách `$1F0` až `$1FF` v okamžiku, kdy procesor začne provádět instrukci NOP na adrese `$549`? Zapište v šestnáctkové soustavě hodnotu každého bytu paměti z rozsahu `$1F0-$1FF`.

Otázka č. 7 (Y)

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

Otázka č. 8 (Y)

Předpokládejte, že pro uvedený počítač Apple II programujeme jednoduchý operační systém s podporou pro vícevláknové programování – náš systém bude podporovat maximálně 16 vláken, všechna vlákna v systému budou součástí jediného procesu, maximální podporovaná hloubka každého volacího zásobníku bude 16 bytů. Pro každé vlákno máme připravený jeden 16 bytový stavový blok (záznam), který obsahuje informace o vlákně (vždy na offsetu `0` od začátku bloku je 1 byte informace o stavu vlákna: `0` = DEAD, `1` = READYTORUN, `2` = RUNNING, `3` = WAITING; na offsetech 1 až 15 si můžete zaznamenat libovolné další potřebné informace). Každé vlákno má přidělené pevné `ID`, které je násobkem 16 (= `0` pro 0. vlákno, `16` pro 1. vlákno, až `240` pro 15. vlákno) – stavový blok pro nějaké vlákno leží na adrese `$200 + ID`. Na adrese `$320` máme uložené `ID` právě běžícího vlákna, na adresu `$310` si můžeme kdykoliv uložit `ID` vlákna, které má běžet po něm.
V systému chceme naimplementovat jednoduchou variantu round-robin plánovače, kdy po nějakém vláknu má jako další běžet vlákno s nejbližším vyšším `ID`, které je připravené. Pokud takové neexistuje, tak se vybere připravené vlákno s nejnižším `ID`. Pokud ani takové neexistuje, aktuální vlákno pokračuje v běhu.
Naprogramujete v Pascalu proceduru `Yield` takového plánovače. Na samotné přepnutí vláken budete potřebovat využít v Pascalovém kódu inline assembler procesoru 6502; mezi Pascalem a assemblerem si informaci o vybraném dalším vlákně předejte v proměnné na adrese `$310`).

Otázka č. 9 (Y)

Předpokládejte, že výše uvedený kód na adresách `$514` až `$54B` byl načten ze spustitelného souboru obvyklého formátu. Chceme, aby náš operační systém umožňoval načtení takového kódu i na jinou bázovou adresu. Je v této situaci vhodné/potřeba, aby spustitelný soubor obsahoval tzv. *relokační tabulku*? Co by pro tento konkrétní kód bylo přesně jejím obsahem? Detailně vysvětlete proč.

Otázka č. 10

Některé procesory mají ve svém příznakovém registru tzv. *supervisor* příznak. Vysvětlete, k čemu tento příznak slouží, a jaký software a proč ho využívá. Nezapomeňte vysvětlit, za jakých okolností procesor hodnotu tohoto příznaku čte, a kdy a jakým způsobem lze hodnotu příznaku měnit.