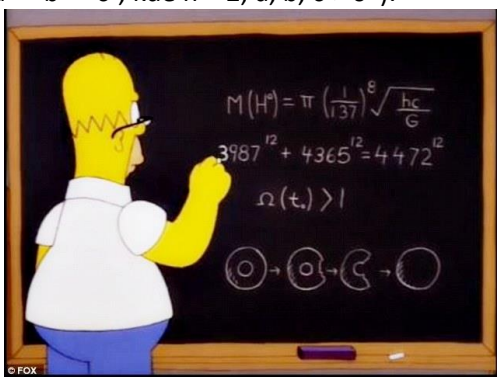
Odpovědi pište na zvláštní odpovědní list s vaším jménem a fotografií. Pokud budete odevzdávat více než jeden list s řešením, tak se na 2. a další listy nezapomeňte podepsat. Do zápatí všech listů vždy napište i/N (kde i je číslo listu, N je celkový počet odevzdaných listů).

Otázka č. 1

Ve 2. díle 10. řady seriálu Simpsonovi se Homerovi podařilo najít příklad čísel, která vyvrací Velkou Fermatovu větu (která zní: „Neexistují přirozená čísla a, b, c, n, pro která platí, že $a^n + b^n = c^n$, kde $n > 2$, $a, b, c \neq 0$“):



Zdálo se nám to divné, a tak jsme se rozhodli, že Homerův příklad ověříme zapsáním jednoduchého programu v Pascalu:

```
program Fermat;
uses Math;
var
  a, b, c, r1, r2 : Single;
begin
  SetExceptionMask([exInvalidOp,
    exDenormalized, exZeroDivide,
    exOverflow, exUnderflow, exPrecision]);
  a := 3987; b := 4365; c := 4472;
  r1 := Power(a, 12) + Power(b, 12);
  r2 := Power(c, 12);
  WriteLn(r1 = r2);
end.
```

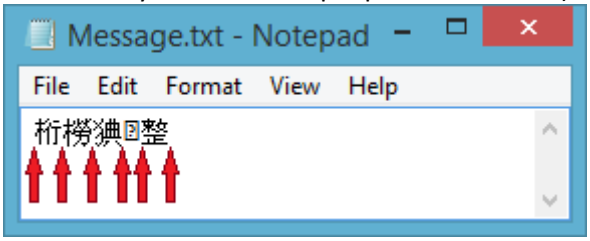
Po spuštění programu jsme zjistili, že program překvapivě opravu vypíše TRUE – bohužel nejlepší matematici na Matfyzu nás přesvědčili, že Velká Fermatova věta opravdu platí, a že v našem programu musíme mít chybu. Detailně vysvětlete, co je v uvedeném programu špatně, a proč tedy nevypisuje správně FALSE (`Single` je v Pascalu 32-bit floating point číslo dle standardu IEEE 754).

Otázka č. 2

Od našeho známého, který používá české Windows, jsme získali textový soubor `Message.txt`, o kterém víme, že je v něm uložený pouze a právě text:

```
Ahoj světe!
```

Když jsme ale soubor otevřeli v Notepadu, zobrazilo se nám následující (červené šipky označují všechna místa, kam jsme pomocí kurzorových kláves schopni posunout kurzor):



Vysvětlete, v jakém „formátu“ je asi uložený původní textový soubor, a proč, resp. z jakého asi důvodu dochází právě k jeho výše uvedenému chybnému zobrazení.

Otázka č. 3

Součástí běžných operačních systémů bývá často tzv. HAL vrstva (Hardware Abstraction Layer). Detailně vysvětlete, k čemu HAL v OS slouží, a jaké má použití HAL výhody oproti systému bez HAL. Zvláště uveďte příklad alespoň jedné funkce, kterou by HAL mohla typicky implementovat, a také vysvětlete, kdo a proč by v takovém systému tuto funkci volal.

Otázka č. 4

Zapište v šestnáctkové soustavě hodnotu 64-bitové proměnné `i` typu `Int64` po provedení níže uvedeného kódu v Pascalu, když víme, že se kód bude překládat pro little endian platformu:

```
i := -10;
i := (
  (
    (
      i and $1122334455667788
    ) xor $5500000000000000
  ) or (
     896 shl 7
  )
) xor $5500000000000000;
```

Otázka č. 5

Předpokládejte, že máme běžný počítač IBM PC kompatibilní, který má na základní desce integrovaný USB 2.0 řadič. Součástí řadiče je i kořenový USB 2.0 hub, jehož 4 porty jsou vyvedeny na zadní stranu počítače. V jednom z těchto portů máme zapojený dedikovaný 3 portový USB 1.1 hub – do jeho 1. portu je zapojena USB 1.1 klávesnice, do jeho 2. portu je zapojen externí USB 2.0 pevný disk používající Advanced Format, a 3. port hubu je zatím volný. Zmíněný pevný disk je již plně nakonfigurovaný a tedy schopný odpovídat na požadavky o čtení/zápis dat jeho sektorů – pro přenos dat implementuje řadič disku standardní USB Mass Storage protokol, kde přečtení dat z disku vyžaduje následující posloupnost kroků:

1) Poslat disku 31 bytovou strukturu Command Block Wrapper (CBW), popisující žádaná data.
2) Přečíst od disku data vyžádaných sektorů, které disk pošle (pozor: nemusí se vejít do jednoho paketu – maximální velikost payloadu jednoho paketu při USB 1.1 přenosu je 256 B).
3) Přečíst 13 bytů struktury Command Status Wrapper (CSW), kterou disk pošle po datech sektorů ve zvláštním nezávislém bloku (jako separátní payload).

Za předpokladu, že je USB disku přidělena adresa 85, a budeme z něj chtít přečíst jeden sektor dat, popište (nakreslete), které všechny pakety budou po USB sběrnici přenášeny po celou dobu přenosu – načrtněte základní strukturu každého paketu, a pro každý paket uveďte, kdo ho vysílá, kdo ho přijímá a velikost payloadu takového paketu.

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 to Accumulator), **STA** ***arg*** (Store from 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, tj. 8-bitová konstanta), ***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 `$614` 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 `$600` 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):

```
$0600 LDA #$05
$0602 PHA
$0603 LDA #$03
$0605 PHA
$0606 JSR $0614
$0609 TAY
$060a TSX
$060b TXA
$060c CLC
$060d ADC #$02
$060f TAX
$0610 TXS
$0611 TYA
$0612 NOP
$0613 NOP
```

```
$0614 TSX
$0615 LDA $0103,X
$0618 BNE $061e
$061a LDA $0104,X
$061d RTS
$061e LDA $0104,X
$0621 PHA
$0622 LDA $0103,X
$0625 SEC
$0626 SBC #$01
$0628 PHA
$0629 JSR $0614
$062c TAY
$062d TSX
$062e TXA
$062f CLC
$0630 ADC #$02
$0632 TAX
$0633 TXS
$0634 TYA
$0635 ASL A
$0636 RTS
```

Otázka č. 6 (X)

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 `$600`, jaký by byl obsah paměti na adresách `$1F0` až `$1FF` v okamžiku, kdy procesor začne provádět instrukci NOP na adrese `$612`? Zapište v šestnáctkové soustavě hodnotu každého bytu paměti z rozsahu `$1F0`-`$1FF`.

Otázka č. 7 (X)

Napište v Pascalu kód funkce (včetně deklarace), která svým chováním bude odpovídat výše uvedené funkci zapsané v assembleru 6502.

Otázka č. 8 (X)

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 `$350` máme uložené `ID` právě běžícího vlákna, na adresu `$351` 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 `$351`).

Otázka č. 9 (X)

Předpokládejte, že výše uvedený kód na adresách `$600` až `$636` 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?

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.