1) Co je floating stav? Čeho se týká?

2) Mějme sériovou linku s 1 start bitem, 7 datovými bity, a 2 stop bity. Jakou musíme zvolit přenosovou rychlost na fyzické úrovni v baudech, abychom dosáhli reálné přenosové rychlosti 65536 bit/s? Pokud zachováme spočítanou přenosovou rychlost na fyzické úrovni, a budeme používat 8 datových bitů místo 7, zvýší se reálná přenosová rychlost datových bitů za sekundu, nebo se naopak sníží?

3) Pomocí osciloskopu jsme zaznamenali průběh napěťových úrovní v průběhu nějakého delšího přenosu dat na datovém vodiči sériové linky RS-232. Rozpoznali jsme, že se při přenosu používají 8-bitové byty, 1 start bit, a 1 stop bit. Dále jsme změřili, že doba trvání jednoho bitu je právě 0,5 ms (milisekundy). Jaká byla reálná přenosová rychlost při zachyceném přenosu, tj. kolik datových bitů se přeneslo za sekundu?

4) Jaký je význam start bitu?

5) Jaký je význam stop bitu?

6) Co je hodinový signál?

7) Nakreslete časový diagram přenosu hodnoty 233 po digitální sériové lince s hodinovým signálem, pokud se jednotlivé bity přenášejí jako MSb-first. Data musí být platná při rostoucí hraně hodinového signálu.

8) Co znamená DDR? Jakou má výhodu?

9) Jak funguje clock recovery?

10) Je RS-232 linka full-duplex nebo half-duplex? Proč?

11) Co je řadič?

12) Co je registr?

13) Co je buffer? Mohla by část datové paměti sloužit jako buffer?

14) Co se stane, pokud přijímající používá jiný komunikační protokol než vysílající, tedy jiný formát paketů?

15) Chceme navrhnout formát paketu, který obsahuje datum ve formě: den/měsíc/rok. Jak má paket vypadat? Kolik bude doopravdy potřeba (bude vhodné) bitů pro reprezentaci jednotlivých částí paketu?

16) Navrhněte formát paketu, kterým chceme přenášet aktuální čas s přesností na milisekundy. Existuje více možností, jak takový formát navrhnout?

17) Mějme sériovou RS-232 linku s 1 start bitem, 8 datovými bity, a 1 stop bitem. Pro vaše formáty paketů z otázek **15** a **16** určete, kolik RS-232 přenosů bude potřeba pro přenesení jednoho takového paketu.

Sada programovacích úloh – společné poznámky: Zatím se nedívejte do zdrojového souboru `SimulatedCpuCommands.py`, jeho obsah si můžete později zkusit naprogramovat, viz úlohy níže pro pokročilé. Pozor, přiložená vzorová řešení nejsou jediná možná, i jiná řešení mohou být správná. Na vzorová řešení se nedívejte, dokud si nezkusíte napsat vlastní implementaci. V žádné úloze zatím neřešíme implementaci funkce „`open()`“ – její implementaci si vyzkoušíte až v nějaké pozdější sadě self-assessment úloh, až se na přednášce dozvíte více informací.

Využijte simulované funkce procesoru `cpu.readDataRegister(`*`identifikátor_řadiče`*`)` a `cpu.readStatusRegister(`*`identifikátor_řadiče`*`)`, obě vracející obsah daného simulovaného 8-bitového registru jako Python číslo `int`.

18) Do projektu `PySimulatedSerialRead-ASSIGNMENT` dopište mezi komentáře `### BEGINNING OF oneByte = serialPort.read() IMPLEMENTION ###` a `### END OF oneByte = serialPort.read() IMPLEMENTION ###` kód, který se bude chovat jako původní „`oneByte = serialPort.read()`“. **Více viz připojené video zadání.**
[Vzorové řešení najdete v projektu `PySimulatedSerialRead`.]

19) Do projektu `PySimulatedSerialMultiread-ASSIGNMENT` dopište mezi komentáře `### BEGINNING OF packet = serialPort.read(4) IMPLEMENTION ###` a `### END OF packet = serialPort.read(4) IMPLEMENTION ###` kód, který se bude chovat jako původní „`packet = serialPort.read(4)`“. Proměnnou `bytesToReceive` berte jako ekvivalent parametru předaného funkci `read`. **Více viz připojené video zadání.**
[Vzorové řešení najdete v projektu `PySimulatedSerialMultiread`.]

Úloha pro středně pokročilé:

20) Do projektu `PySimulatedSerialReadWithTimeout-ASSIGNMENT` opět dopište implementaci „`packet = serialPort.read(4)`“ jako v přechozím příkladu. Nicméně nyní má vaše implementace brát v potaz hodnotu nastavenou do proměnné `timeout` – pokud od posledního přijatého datového bytu uplynul část `timeout` v sekundách, tak má kód „`packet = serialPort.read(4)`“ vrátit všechny dosud přijeté datové byty, i když jejich počet ještě nedosáhl 4. Tedy na vstupních datech, tak jak jsou uvedena v projektu, tak má vrátit jen 3 hodnoty 96, 44, 28, a nikoliv už poslední 0.

Pokud je neznáte, tak v Pythonu existuje typ `datetime` reprezentující nějaký bod v čase (datum a čas), a typ `timedelta` reprezentující nějaký časový interval. S typy můžete pracovat pomocí:

```
from datetime import timedelta
from datetime import datetime
```

Funkce `datetime.now()` vrací datetime objekt reprezentující aktuální datum a čas. Operace rozdílu (`-`) mezi dvěma hodnotami `datetime` vrací rozdíl v časech jako `timedelta`. Na objekty `timedelta` i `datetime` lze aplikovat operátory rovnosti a nerovnosti (`==`, `!=`, `<`, `>`, atd.). Object `timedelta` pro daný počet sekund lze zkonstruovat pomocí `timedelta(seconds=počet_sekund)`.
[Vzorové řešení najdete v projektu `PySimulatedSerialReadWithTimeout`.]

Úlohy pro pokročilé (je třeba znalost implementace vlastních funkcí [viz pozdější přednášky z Programování I – tj. *všichni se k těmto úlohám můžou vrátit později*], a rozdíl mezi globálními a lokálním proměnnými):

Upozornění: častá chyba je, že chcete použít globální proměnnou, ale neoznačíte její jméno pomocí `global` na začátku funkce a pak omylem zakládáte novou lokální proměnnou platnou jen uvnitř funkce, viz:

Následující Python program:

```
abc = 42

def myFunction1():
    abc = 111
    print("myFunction1: variable 'abc' contains:", abc)

def myFunction2():
    global abc
    abc = 222
    print("myFunction2: variable 'abc' contains:", abc)

print("Main program: variable 'abc' contains:", abc)
print("Main program: calling myFunction1 ...")
myFunction1()
print("Main program: variable 'abc' contains:", abc)
print("Main program: calling myFunction2 ...")
myFunction2()
print("Main program: variable 'abc' contains:", abc)
```

po spuštění vypíše následují text na konzoli:

```
Main program: variable 'abc' contains: 42
Main program: calling myFunction1 ...
myFunction1: variable 'abc' contains: 111
Main program: variable 'abc' contains: 42
Main program: calling myFunction2 ...
myFunction2: variable 'abc' contains: 222
Main program: variable 'abc' contains: 222
```

Tedy si všimněte, že funkce `myFunction1` nezměnila obsah globální proměnné `abc`, ale funkce `myFunction2` ano.

21) Do projektu `PySimulatedSerialRead-CPU-COMMANDS-ASSIGNMENT` (projekt v souboru `PySimulatedSerialRead.py` obsahuje již vzorové řešení z úlohy `PySimulatedSerialRead`) do zdrojového

souboru `SimulatedCpuCommands.py` doimplementujte simulaci chování příkazů CPU pro čtení obsahu datového a stavového registru simulovaného řadiče RS-232 linky. Jako simulovaný zdroj přijímaných dat použijte globální proměnnou `inputData` – hodnota `-1` reprezentuje, že k příjmu žádných nových dat nedošlo, nezáporná hodnota reprezentuje přijatý datový byte. **Pokud řešení nedokážete vymyslet, tak se zkuste podívat na doplňkové video pro pokročilé.**
[Vzorové řešení najdete v souboru `SimulatedCpuCommands.py` např. v projektu `PySimulatedSerialRead`.]

22) Do projektu `PySimulatedSerialReadWithTimeout-CPU-COMMANDS-ASSIGNMENT` (projekt v souboru `PySimulatedSerialReadWithTimeout.py` obsahuje již vzorové řešení z úlohy `PySimulatedSerialReadWithTimeout`) do zdrojového souboru `SimulatedCpuCommands.py` doimplementujte simulaci chování příkazů CPU pro čtení obsahu datového a stavového registru simulovaného řadiče RS-232 linky stejně jako v předchozí úloze. Nyní ale simulovaná vstupní data čerpejte z proměnné `inputEvents` (seznam dvojic, kde nultá položka dvojice je objekt `timedelta`, a první položka je hodnota přijatého datového bytu) – hodnota datového bytu se má v datovém registru simulovaného řadiče RS-232 objevit až po době (od začátku programu) určené nultým parametrem události. Předpokládejte, že události jsou v proměnné `inputEvents` vždy uspořádány podle rostoucího času jejich očekávaného vzniku od spuštění programu.
[Vzorové řešení najdete v souboru `SimulatedCpuCommands.py` např. v projektu `PySimulatedSerialReadWithTimeout`.]

Poznámka: Cílem těchto úloh je, abyste si doma v rámci opakování látky z přednášky mohli sami ověřit, jak jste látce porozuměli. Úlohy jsou koncipovány víceméně přímočaře, a po projití poznámek a případně video záznamů z přednášek by jejich řešení mělo být zřejmé. Pro řešení úloh tedy není potřeba studium látky nad rámec probraný na přednáškách (nicméně je třeba mít i znalosti a pochopení látky z paralelně probíhajících přednášek a cvičení z předmětu Programování I). Pokud i po detailním a opakovaném projití látky z přednášek máte s řešením těchto úloh problém, tak se na nejasnosti co nejdříve ptejte na on-line konzultaci k přednášce, případně zvažte se mnou domluvit na konzultaci (zvlášť pokud tento stav u vás přetrvává i po dalších přednáškách).

Upozornění: Úlohy jsou vybrány a postaveny tak, abyste si po přednášce a před přednáškou následující mohli rychle připomenout hlavní části probrané látky a ověřit si její pochopení. Nicméně úlohy nejsou vyčerpávajícím přehledem látky z přednášek a tady rozhodně nepokrývají kompletní látku přednášek, která bude vyžadována u zkoušky. Pokud tedy u každé úlohy víte, jak by se měla řešit, tak to ještě neznamená, že jste dostatečně připraveni na zkoušku – nicméně jste jistě na velmi dobré cestě. Každopádně nezapomeňte, že na zkoušce se vyžaduje pochopení a porozumění právě všem konceptům ze všech přednášek, a navíc jsou zkouškové příklady postaveny komplexněji tak, aby ověřily také vaši schopnost přemýšlet nad látkou napříč jednotlivými přednáškami.