1) V jakých jednotkách typicky měříme rychlost instrukce na nějaké procesorové architektuře?

2) Předpokládejte 2 modely CPU implementující stejnou instrukční sadu a přesně stejnou architekturu. Pokud by nás zajímalo, kolik *ns* (nanosekund) bude trvat nějaká konkrétní instrukce, tak jaké hlavní parametry CPU tuto dobu ovlivňují?

3) Jaké operace budou typicky spadat do množiny nejrychlejších instrukcí, které procesor podporuje?

4) Předpokládejte níže uvedené příklady instrukcí procesoru řady x86 (použijte slidy 50 až 55 z prezentace 8. přednášky jako referenci k základním instrukcím procesorové architektury x86). Proveďte odhad rychlosti uvedených instrukcí a seřaďte je do kategorií od nejrychlejší po nejpomalejší:

a) `MOV EAX, [00A01580h]`
b) `ADD EAX, EBX`
c) `SHL EAX, 1`
d) `MOV EAX, EBX`
e) `ADD EAX, [00A01591h]`
f) `ADD EAX, FF650001h`
g) `MOV [00A01584h], EAX`
h) `MOV EBX, EAX`

5) **Pro lehce pokročilé** (je třeba si propojit znalosti z 9. přednášky s přednáškami dřívějšími): Vraťte se k otázce 4, a předpokládejte, že máme dva modely 32-bitového CPU řady x86 – model A s operační pamětí komunikuje výhradně pomocí protokolu s 32-bitovými slovy (tj. k CPU modelu A lze připojit pouze 32-bitové paměti DRAM); model B s operační pamětí komunikuje výhradně pomocí protokolu s 16-bitovými slovy (tj. k CPU modelu B lze připojit pouze 16-bitové paměti DRAM). Pokud jsou ostatní parametry modelů A a B shodné, tak bude se nějak lišit rychlost vykonávání instrukcí z otázky 4? A pokud ano, tak které instrukce z a) až h) budou na modelu B rychlejší, resp. pomalejší než na modelu A?

6) Jak jsou přesně v Pythonu reprezentována celá čísla?

7) **A)** Kolik bytů budou v Pythonu zabírat následující hodnoty typu `int`?
B) Jakou nejmenší velikost proměnné bychom pro každou z těchto hodnot mohli vybrat v jazyce C nebo C# (předpokládejte pro jednoduchost existenci stejných celočíselných číselných typů jako v knihovně `numpy`)?

a) `512`
b) `16000000`
c) `4000000000`
d) `64000000000`
e) `128`
f) `-128`
g) `-130`

8) Co je tzv. aritmetické přetečení, resp. aritmetické podtečení?

9) Jaké jsou výhody Pythonové implementace celých čísel? A jaké jsou její nevýhody?

10) Jaké jsou výhody Pythonové beztypovosti proměnných? A jaké jsou její nevýhody?

11) Pokud procesor podporuje instrukci násobení a instrukci dělení, tak jakou výkonost byste u nich očekávali ve srovnání s instrukcí sčítání?

12) Jaký je rozdíl mezi logickým a aritmetickým posunem doprava?

13) Předpokládejte níže uvedené operace na unsigned proměnných `A` a `B` typu `uint32`, a signed proměnné `C` a `D` typu `int32` (z knihovny `numpy`). **Pokud je to přímočaře možné**, tak každý z výrazů přepište tak, abyste operace násobení, resp. dělení přepsali do ekvivalentního výrazu pomocí bitových operací:

a) `A * 256`
b) `B / 32`
c) `A / B`
d) `C * 8`

e) `D / 16`

14) Zapište níže uvedená reálná čísla jako posloupnost bitů (zleva od MSb doprava směrem k LSb), pokud číslo budeme reprezentovat ve dvojkové soustavě dle dané fixed-point reprezentace (s pevnou řádovou čárkou):

a) `14,75` jako fixed-point 5.3
b) `14,75` jako fixed-point 4.4
c) `270,625` jako fixed-point 10.6
d) `270,625` jako fixed-point 22.2
e) `511,5` jako fixed-point 10.6
f) `254` jako fixed-point 8.8

15) Předpokládejte, že v paměti jsou na adresách `$0010F300` až `$0010F31F` uloženy zatím samé nulové byty. Dále předpokládejte, že budeme níže uvedená reální čísla reprezentovat ve dvojkové soustavě dle dané fixed-point reprezentace (s pevnou řádovou čárkou). Poté tyto fixed-point hodnoty uložíme vždy na uvedenou adresu do zmíněné paměti. Napište hexdump (definice a vysvětlení viz zadání self-assessment úloh pro 7. přednášku) finální stavu paměti mezi adresami `$0010F300` až `$0010F31F` po uložení všech uvedených hodnot. Hodnoty se ukládají v **Little Endian** pořadí:

a) `18,375` jako fixed-point 8.24 na adrese `$0010F300`
b) `18,375` jako fixed-point 12.20 na adrese `$0010F308`
c) `1040,5` jako fixed-point 8.8 na adrese `$0010F30C`
d) `1,5` jako fixed-point 4.4 na adrese `$0010F310`
e) `65535` jako fixed-point 24.8 na adrese `$0010F314`

16) Jaká je hlavní výhoda fixed point reprezentace?

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.