1) Co je *příznak*? A co je *příznakový registr*? Mohou být příznaky pouze v příznakovém registru?

2) Jaké jsou příklady situací, kdy CPU mění obsah nějakého příznaku v příznakovém registru?

3) Co je *akumulátorová architektura* CPU?

Poznámka: V dále uvedených úlohách použijte slidy 12 až 28 z prezentace 7. přednášky + slidy 9 až 42 z prezentace 8. přednášky jako referenci k základním instrukcím nepodmíněného skoku, load, a store, a bitovým a aritmetickým operacím na procesoru 6502. Pro zápis v assembleru využívejte konvenci assembleru pro 6502 uvedenou na slidech.

4) Máme níže uvedené příkazy v jazyce C (význam operátorů je pro uvedené příklady stejný jako v jazyce Python), které používají **32-bitové** proměnné `one` (leží na adrese `0xA400`) a `two` (leží na adrese `0xA404`). Tyto příkazy přímočaře přeložíme do strojového kódu procesoru 6502. Zapište, jak bude daný příkaz vypadat v assembleru 6502:

a) `two = two & one;`
b) `one = ~two;`

5) Pro uvedené dvojice hodnot A, B zapsaných ve dvojkové soustavě napište hodnotu součtu A+B (výsledek zapište také ve dvojkové soustavě + zapište pro každou číslici i hodnotu přenosu do vyššího řádu = carry):

a) A = `11101010`, B = `11011111`,
b) A = `11101011`, B = `101`.

6) Co je obecně vstupem a co výstupem 32-bitového sčítání s přenosem?

7) Máme níže uvedené příkazy v jazyce C, které používají **32-bitové unsigned** proměnné `one` (leží na adrese `0xA400`) a `two` (leží na adrese `0xA404`), a **16-bitovou unsigned** proměnnou `three` (leží na adrese `0xA408`). Tyto příkazy přímočaře přeložíme do strojového kódu procesoru 6502. Zapište, jak bude daný příkaz vypadat v assembleru 6502:

a) `two = one + two;`
b) `one = two + 509;`
c) `two = one + three;`

8) Máme níže uvedené příkazy v jazyce C, které používají **32-bitové signed** proměnné `one` (leží na adrese `0xA400`) a `two` (leží na adrese `0xA404`), a **16-bitovou signed** proměnnou `three` (leží na adrese `0xA408`). Tyto příkazy přímočaře přeložíme do strojového kódu procesoru 6502. Zapište, jak bude daný příkaz vypadat v assembleru 6502:

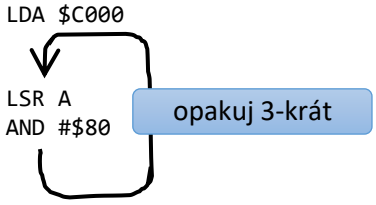
a) `two = one + two;`
b) `one = two + 509;`

Pozor, správné řešení varianty c) není úplně triviální, je třeba ho trochu promyslet. Pro dočasné proměnné můžeme použít paměť mezi adresami `0xD500` a `0xD600`. Při řešení můžete brát v potaz nezjednodušené chování operací bitových posunů, viz bonusový slide 49 z prezentace 8. přednášky.

c) `two = one + three;`

Pokud byste ve svém kódu varianty c) potřebovali zopakovat nějakou posloupnost instrukcí, tak je nemusíte do řešení kopírovat, ale stačí naznačit opakování takové posloupnosti – tedy např. místo kódu vlevo (viz níže) by do řešení postačovalo napsat kód vpravo (viz níže):

```
LDA $C000
LSR A
AND #$80
LSR A
AND #$80
LSR A
AND #$80
```



9) Pro uvedené dvojice hodnot A, B zapsaných ve dvojkové soustavě napište hodnotu rozdílu A-B. Při výpočtu postupujte podle algoritmu subtract with borrow (výsledek zapište také ve dvojkové soustavě + zapište pro každou číslici i hodnotu vypůčení z vyššího řádu = borrow). Rozdíl proveďte v 8-bitové přesnosti:

a) A = `11101010`, B = `11011111`,
b) A = `11101011`, B = `101`,
c) A = `1101`, B = `11100`.

10) Co je obecně vstupem a co výstupem 32-bitového rozdílu ve variantě subtract with borrow?

11) Jaký je rozdíl mezi *subtract with borrow* a *subtract with carry*? Jaké jsou jejich výhody a nevýhody?

12) Máme níže uvedené příkazy v jazyce C, které používají **32-bitové unsigned** proměnné `one` (leží na adrese `0xA400`) a `two` (leží na adrese `0xA404`), a **16-bitovou unsigned** proměnnou `three` (leží na adrese `0xA408`). Tyto příkazy přímočaře přeložíme do strojového kódu procesoru 6502. Zapište, jak bude daný příkaz vypadat v assembleru 6502:

a) `two = one - two;`
b) `one = two - 509;`
c) `two = one - three;`

Poznámka: V dále uvedených úlohách 13 a 15 použijte slidy 50 až 55 z prezentace 8. přednášky jako referenci k základním instrukcím procesorové architektury x86. Pro zápis v assembleru využívejte konvenci assembleru pro x86 uvedenou na slidech.

13 [x86]) Máme níže uvedené příkazy v jazyce C, které používají **32-bitové unsigned** proměnné `one` (leží na adrese `0x0000A400`) a `two` (leží na adrese `0x0000A404`), a **64-bitové unsigned** proměnné `three` (leží na adrese `0x0000A408`) a `four` (leží na adrese `0x0000A410`). Tyto příkazy přímočaře přeložíme do strojového kódu procesoru x86. Zapište, jak bude daný příkaz vypadat v assembleru x86:

a) `two = one + two;`
b) `three = four + three;`
c) `four = one - three;`

14 [6502]) Máme níže uvedené příkazy v jazyce C, které používají **8-bitové unsigned** proměnné `A` (leží na adrese `0xC12A`), `B` (leží na adrese `0xC12B`) a `C` (leží na adrese `0xC12C`), a **16-bitové unsigned** proměnné `D` (leží na adrese `0xC200`) a `E` (leží na adrese `0xC202`). Pro dočasné proměnné můžeme použít paměť mezi adresami `0xD500` a `0xD600`. Tyto příkazy přímočaře přeložíme do strojového kódu procesoru 6502. Zapište, jak bude daný příkaz vypadat v assembleru 6502:

a) `A = A + B + C;`
b) `A = (A - B) + (A – C);`
c) `E = E + D + 15 + (B – A);`

15 [x86]) Máme níže uvedené příkazy v jazyce C, které používají **8-bitové unsigned** proměnné `A` (leží na adrese `0x1000C12A`), `B` (leží na adrese `0x1000C12B`) a `C` (leží na adrese `0x1000C12C`), a **16-bitové unsigned** proměnné `D` (leží na adrese `0x1000C200`) a `E` (leží na adrese `0x1000C202`). Pro dočasné proměnné můžeme použít paměť mezi adresami `0x1000D500` a `0x1000D600`. Tyto příkazy přímočaře přeložíme do strojového kódu procesoru x86. Zapište, jak bude daný příkaz vypadat v assembleru x86:

a) `A = A + B + C;`
b) `A = (A - B) + (A – C);`
c) `E = E + D + 15 + (B – A);`

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.