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).

```
00020000: 48 45 4C 4C 4F 20 20 20|54 58 54 20 00 6F 29 5E  | HELLO   TXT .o)^
00020010: 93 43 93 43 00 00 37 8F|76 43 02 00 0E 00 00 00  | .C.C..7.vC......
00020020: 53 45 43 4F 4E 44 20 20|54 58 54 20 00 39 2B 5E  | SECOND  TXT .9+^
00020030: 93 43 93 43 00 00 3D 5E|93 43 03 00 38 0E 00 00  | .C.C..=^.C..8...
00020040: 54 48 49 52 44 20 20 20|54 58 54 20 00 14 2E 5E  | THIRD   TXT ...^
00020050: 93 43 93 43 00 00 85 91|76 43 08 00 0A 03 00 00  | .C.C....vC......
```

Společná část pro otázky označené X

Operační systémy Windows i Linux mimo jiné podporují disky naformátované souborovým systémem FAT. Tento souborový systém vychází ze 70. a 80. let, a má tedy různá v té době běžná omezení – např. nepodporování mezer ve jménech a příponách souborů, nebo limit na maximálně 8 znaků jména souboru a maximálně 3 znaky jeho přípony. Předpokládejte, že disk 1 naformátovaný variantou FAT souborového systému jsme od boot sektoru až po poslední sektor nakopírovali do souboru `FAT16DiskImage.bin` uloženého na jiném disku 2 – tím jsme vytvořili tzv. *obraz disku* 1 (disk image). Část tohoto obrazu disku jsme si zobrazili v hex vieweru (viz záhlaví zadání výše) – zobrazená část ukazuje od začátku tzv. *kořenový adresář* (*root directory*), což je část metadat souborového systému popisující informace o souborech, které ve jméně nemají prefix žádné cesty. Popis binární struktury kořenového adresáře najdete **v příloze**.

Otázka č. 1 (X)

Pro zobrazený obraz disku 1 napište v desítkové soustavě datum a čas poslední modifikace (vytvoření) souboru `SECOND.TXT`. Vysvětlete, jak jste k danému výsledku došli.

Otázka č. 2 (X)

Napište v Pythonu implementaci funkce:

```
def PrintFileNames(
    diskImageFileName : str, rootDirOffset : int,
    rootDirLen : int):
```

která jako své argumenty bere jméno souboru s obrazem disku naformátovaného souborovým systémem FAT, 2. argument je offset od začátku souboru, kde se nachází začátek metadat uložených v kořenovém adresáři, 3. argument je délka metadat z kořenového adresáře v bytech. Pro jednoduchost nerealisticky předpokládejte, že funkci jsou předávány pouze korektní hodnoty argumentů, tj. není třeba kontrolovat jejich správnost (stejný předpoklad pak použijte i při řešení **otázky 3**).
Funkci naprogramujte tak, aby vypsala jména všech souborů uložených v kořenovém adresáři. Každé jméno se má vypsat na zvláštní řádek, má být zapsané v hranatých závorkách, a má vypadat tak, jak by ho zapsal uživatel systému (tedy s tečkou a bez přebytečných mezer) – např. pro zobrazený hex view by vaše funkce zobrazila:

```
[HELLO.TXT]
[SECOND.TXT]
[THIRD.TXT]
```

Otázka č. 3 (X)

Napište v Pythonu implementaci funkce:

```
def PrintModificationDateTime(
    diskImageFileName : str, rootDirOffset : int,
    rootDirLen : int):
```

Funkce má stejný význam parametrů jako **v otázce 2**, a má i opět zobrazit informace o všech souborech v kořenovém adresáři. Funkce má ale zobrazit datum a čas modifikace každého souboru v následujícím formátu (pro každý soubor na zvláštní řádek) – viz následující dva příklady:

```
1.5.1980 12:34
26.4.1986 22:44
```

Otázka č. 4

Předpokládejte, že chceme reálné číslo `-1029,25` uložit do paměti od adresy `0x0A0231C0` jako typ `single`, tj. 32-bitové floating-point číslo dle standardu IEEE 754, tj. mantisa je normalizována se skrytou 1 a zabírá spodních 23 bitů, pak následuje 8-bitový exponent uložený ve formátu s posunem [bias] +127, a poslední bit, tedy `MSb`, je znaménkový bit. Napište v šestnáctkové soustavě hodnotu každého bytu paměti, ve kterém bude uložená nějaká část této hodnoty – předpokládejte, že pracujeme na 32-bitovém **little-endian** CPU.

Otázka č. 5

Předpokládejte, že v typické implementaci Pythonu běžící na 32-bitovém počítači provedeme následující příkaz:

```
a = 300
```

Nakreslete a detailně vysvětlete, jak bude vypadat veškerá paměť zabraná touto proměnnou a její hodnotou. Pro jednotlivé části dat napište očekávané velikosti.
Pokud poté provedeme ještě příkaz:

```
a += 1
```

tak opět detailně vysvětlete, jak se která část paměti změní a jaký bude její obsah.

Otázka č. 6

Detailně vysvětlete rozdíly (a výhody a nevýhody) mezi typickými variantami tzv. *volatile* pamětí. Proč a v jakém kontextu takové *volatile* paměti v počítači typicky využíváme? Proč místo nich nepoužíváme nějaké *non-volatile* paměti?

Otázka č. 7

Jaké signály najdeme v klasickém 4-pinovém USB konektoru? Vysvětlete, jaký je jejich význam. Dále v tomto kontextu vysvětlete koncept tzv. *clock recovery*, a popište jaký má vztah k počtu a druhu signálů USB konektoru.

Společná část pro otázky označené Y

Předpokládejte níže popsaný **32-bitový little-endian** CPU s obecnou registrovou architekturou vycházející z architektury x86 (IA-32) – jedná se o procesor s 32-bitovým adresovým prostorem. Procesor má mimo jiné obecné registry `EAX`, `EBX`, `ECX`, `EDX` (pro každý registr existuje i pohled na jeho spodních 16 bitů pod jmény registrů `AX`, `BX`, `CX`, `DX`, a stejně tak i pohled na jeho spodních 8 bitů pod jmény `AL`, `BL`, `CL`, `DL`), příznakový registr `EFLAGS` s běžnými příznaky, a registr `EIP` (instruction pointer). V instrukční sadě jsou mimo jiné i následující instrukce (příznakový registr modifikují pouze aritmetické operace, ale instrukce přenosu dat nikoliv) – každá z uvedených instrukcí má 32-bit, 16-bit a 8-bitovou variantu, pro konkrétní variantu instrukce jsou **oba** její **operandy vždy** o **stejné bitové přesnosti** (tedy např. 32-bitová varianta má všechny operandy 32-bitové, apod.; variantu instrukce volíme druhem zdrojového/cílového registru):

`MOV reg,imm/[addr]`	(load register)
`MOV [addr],reg`	(store register)
`MOV reg0,reg1`	(copy from `reg1` to `reg0`)
`ADD reg,imm/[addr]/reg`	(add without carry)
`ADC reg,imm/[addr]/reg`	(add with carry)
`SUB reg,imm/[addr]/reg`	(subtract without carry)
`SBB reg,imm/[addr]/reg`	(subtract with borrow)
`CLC`	(clear carry)
`STC`	(set carry)
`OR reg,imm/[addr]/reg`	(bitwise OR)
`AND reg,imm/[addr]/reg`	(bitwise AND)
`SHR reg,imm/CL`	(logical shift right)
`SHL reg,imm/CL`	(logical shift left)

Všechny výše uvedené instrukce se dvěma operandy mají vždy **vlevo cílový** a **vpravo zdrojový** operand. Instrukce mohu mít pro každou „bitovost“ ještě následující varianty operandů (povolené varianty viz definice konkrétní instrukce):

- immediate hodnota `imm`
- absolutní adresa `[addr]`, kde `[addr]` může být:
  - `[imm]` adresa daná konstantou `imm`
  - `[reg32]` adresa daná obsahem 32-bit registru `reg32`
  - `[reg32 +/- imm]` adresa daná součtem/rozdílem obsahu 32-bit registru `reg32` a konstanty `imm`
- libovolný registr `reg`

Otázka č. 8 (Y)

Předpokládejte, že v programu v jazyce C# máme deklarované 4 proměnné `a`, `b`, `c`, `d` typu `ushort`, který je ekvivalentní typu `uint16` z balíčku `numpy`. Víme, že pro proměnné překladač vyhradil místo hned za sebou v pořadí `a`, `b`, `c`, `d` směrem k vyšším adresám, kde první proměnná `a` leží na adrese `0x00051000`. Dále víme, že pro dočasné proměnné můžeme využít libovolnou paměť mezi adresami `0x7F000000` a `0x7F00FFFF`. Zapište v assembleru výše uvedeného procesoru, jak by se přeložil níže uvedený C# výraz (neprovádějte žádné optimalizace a algebraická zjednodušení [jako odstraňování závorek]):

```
a = a - (c + d) - (b + d)
```

Otázka č. 9 (Y)

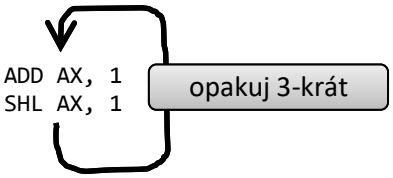
Typický procesor bude mít ve své instrukční sadě i instrukci pro znaménkové rozšíření – nicméně nyní předpokládejte, že výše uvedený procesor takovou instrukci nemá. Zapište pouze s využitím výše uvedených instrukcí kód v assembleru, který očekává, že ve spodních 8 bitech registru AX je 8-bitové znaménkové číslo (obsah horních 8 bitů není definovaný), a po jehož dokončení bude v registru BX 16-bitová hodnota jako znaménkové rozšíření zdrojové 8-bitové hodnoty.

Pokud byste ve svém kódu potřebovali zopakovat nějakou posloupnost instrukcí, tak je nemusíte do řešení kopírovat, ale stačí naznačit opakování takové posloupnosti – tedy např. místo:

```
MOV AX, [000051A0h]
ADD AX, 1
SHL AX, 1
ADD AX, 1
SHL AX, 1
ADD AX, 1
SHL AX, 1
```

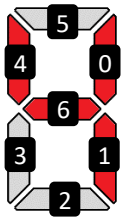
by do řešení postačovalo napsat:

```
MOV AX, [000051A0h]
```



Otázka č. 10

Předpokládejte, že máme k dispozici řadič typu `TPIC2810` od společnosti Texas Instruments sloužící k ovládání zobrazení na 7-segmentovém LED displeji. S řadičem se komunikuje pomocí standardní varianty sběrnice I^2C (všechny byty jsou posílány jako MSb-first). Datasheet tohoto řadiče najdete **v příloze**. Předpokládejte, že k řadiči `TPIC2810` máme na jeho výstupy `DRAIN0` (odpovídá bitu `D0`), až `DRAIN6` (odpovídá bitu `D6`) připojen jeden 7-segmentový LED displej – viz obrázek níže, kde je pro každý segment označeno, na jaké číslo výstupu je daný segment připojený. Za předpokladu, že na displeji chceme rozsvítit symbol 4, viz obrázek (červená = svítící segmenty = výstup na `1`, šedá = zhasnuté segmenty = výstup na `0`):



tak, napište v šestnáctkové soustavě hodnoty **všech** bytů (bez ACK bitu), které se budou přenášet po I^2C sběrnici, pokud chceme zařídit, aby stav výstupů řadiče odpovídal právě uvedenému obrázku zobrazení LED displeje. Předpokládejte, že piny `A0`, `A1`, `A2` řadiče `TPIC2810` jsou připojeny na zem (GND).