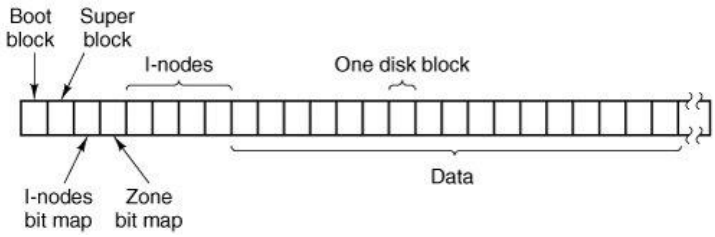
Odpovědi pište na zvláštní odpovědní list s vaším jménem a fotografií. Pokud budete odevzdávat více než jeden list s řešením, tak se na 2. a další listy nezapomeňte podepsat. Do zápatí všech listů vždy napište i/N (kde i je číslo listu, N je celkový počet odevzdaných listů).

Společná část pro otázky označené X

Předpokládejte, že máme soubor (viz jeho hexdump/hexview **v příloze**), který reprezentuje obraz disku naformátovaného souborovým systémem *Minix* (tj. soubor v sobě obsahuje uložená za sebou obsah všech sektorů, které tento souborový systém zabírá). Minix filesystem si logicky disk dělí na tzv. bloky (block) nebo-li zones, kde 1 blok má velikost 1 KiB (tj. dva 512 bytové sektory). Blok číslo 0 obsahuje boot sektor, blok číslo 1 je tzv. superblock a obsahuje základní informace o naformátovaném souborovém systému. Informace o volných blocích je uložena v tzv. bitmapě volných zón. Informace o jednom souboru jsou uložené v datové struktuře nazvané inode – inody jsou číslované od 1. Každému souboru je přiřazený právě jeden inode. Celkový počet inodů je daný při formátování souborového systému – tj. tím je daný i maximální počet souborů, které můžeme do souborového systému uložit. Všechny inody jsou uložené za sebou v tabulce inodů (bloky označené I-nodes na obrázku níže). Informace o volných inodech je uložena v tzv. bitmapě volných inodů (I-nodes bitmap).
Adresářové soubory obsahují seznam dvojic – jméno souboru a odkaz na jeho inode (číslo inodu). Celková struktura disku naformátovaného Minix file systémem je následující:



Detailnější specifikace metadat souborového systému je **v příloze**. Veškerá metadata souborového systému používají pořadí bytů **little-endian**.
V přiloženém obrazu disku začíná tabulka inodů od offsetu `$1000`, první datový blok má číslo `$1A` a tedy začíná od offsetu `$6800`, data kořenového adresáře jsou uložena v bloku `$1A` (všimně te si, že kořenový adresář obsahuje dva záznamy „tečka“ a „dvě tečky“, které se oba odkazují na inode číslo 1, tedy na inode samotného kořenového adresáře).

Otázka č. 1 (X)

Zapište jako číslo v desítkové soustavě, kolik je na disku celkem (i nepoužitých) inodů (number of inodes).

Otázka č. 2 (X)

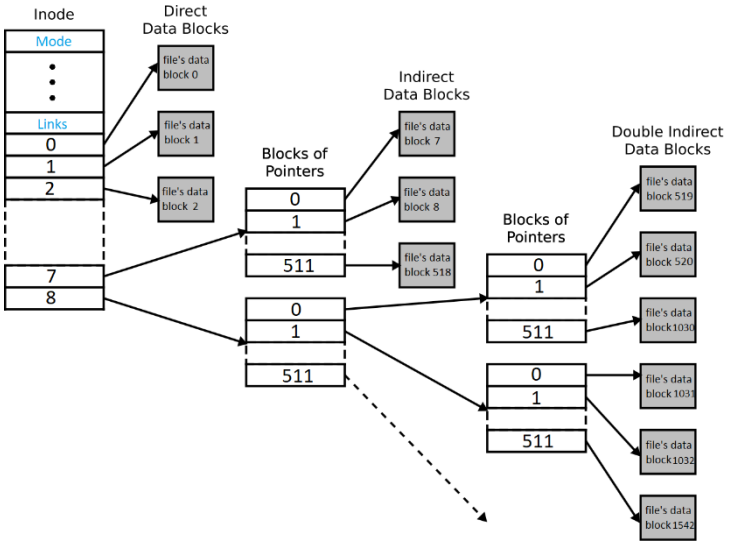
Zapište jako číslo v šestnáctkové soustavě, jako bude vypadat položka MODE v inodu souboru, který má nastavené právě následující příznaky: je „*regular file*“; jeho uživatel (user) do něj smí zapisovat, i z něj smí číst; skupina (*group*) smí ze souboru pouze číst; ostatní (*world/other*) k souboru nemají mít žádný přístup. Pozor, konstanty uvedené ve specifikaci v části s *#define* příkazy jsou uvedené v osmičkové soustavě – **viz příloha** o osmičkové soustavě z wikipedie.

Otázka č. 3 (X)

Naprogramujte v Pascalu následující proceduru:

```
procedure PrintBlocks(name : string; iOfs : longword);
```

která jako argument `name` získá jméno souboru s obrazem disku naformátovaného Minix filesystemem, a dále v argumentu `iOfs` offset od začátku obrazu (souboru s obrazem), kde se nachází inode nějakého souboru uloženého ve filesystemu. Procedura má vypsat čísla všech bloků (zón), které obsahují samotná data souboru (jedno číslo na jeden řádek). Pozor, že číslo `ZONE7` obsahuje číslo bloku, který neobsahuje data samotného souboru, ale teprve čísla bloků se samotnými daty souboru, podobně pro `ZONE8`, ale dvojúrovňově – viz specifikace části X, a obrázek níže:



Otázka č. 4 (X)

Soubor `textovýsoubor.txt` v sobě obsahuje text „`Slunce …`“. Rozhodněte a detailně vysvětlete, jaké kódování se asi používá v názvech souborů, a jaké kódování je asi použito pro reprezentaci samotného textu v uvedeném souboru.

Otázka č. 5 (X)

Jaká je délka souboru `first.txt` v bytech? A jaký je jeho obsah? Jak se pozná, že jeho obsahem jsou právě byty, které jste uvedli? Vše detailně vysvětlete.

Obrázek Minix FS převzat z:
https://flylib.com/books/en/3.275.1.54/1/
Obrázek zone pointerů převzat a upraven z:
https://en.wikipedia.org/wiki/Inode_pointer_structure#/media/File:Ext2-inode.svg
(licence CC BY-SA 4.0)

Otázka č. 6 (X)

Předpokládejte, že v globální proměnné `bitmap`:

```
var
    bitmap : array[0..1023] of byte;
```

máme načtený obsah jednoho bloku s „bitmapou“ volných zón. Pokud je určitý blok (zóna) volný, tak v jemu přiděleném bitu v bitmapě je `0`, pokud je blok zabraný, je v daném bitu `1`. LSb nultého bytu je bit reprezentující stav bloku `0`, MSb nultého bytu je bit bloku `7`, LSb prvního bytu je bit bloku `8`, MSb prvního bytu je bit bloku `15`, atd. Napište hlavní program tak, aby vypsal číslo bloku, kde začíná nejdelší souvislá posloupnost volných bloků. Tedy např. pro data:

`67 F0 F7 FF ... FF`

by program vypsal hodnotu `7`. Pro své řešení vhodně využijte bitové operace.

Otázka č. 7

Detailně vysvětlete, co to je tzv. hardwarové přerušení (IRQ), jak vzniká, a jak probíhá jeho obsluha, tj. jak přesně typický moderní desktopový procesor (např. architektury x86) reaguje na příjem IRQ.

Otázka č. 8

Předpokládejte, že chceme reálné číslo `-2046,25` uložit do Pascalové proměnné ve standardním formátu double. Typ `double` je 64-bitové floating-point číslo dle standardu IEEE 754, tj. mantisa je normalizována se skrytou 1 a zabírá spodních 52 bitů, pak následuje 11-bitový exponent uložený ve formátu s posunem [bias] +1023, a poslední bit, tedy MSb, je znaménkový bit. Nyní program obsahující takovou proměnnou spustíme na počítači s 32-bitovým little-endian CPU, do proměnné uložíme uvedenou hodnotu `-2046,25`, a zjistili jsme, že je proměnná uložená na adrese `0x0A0867D0`. Napište v šestnáctkové soustavě hodnotu každého bytu paměti, ve kterém bude uložená nějaká část proměnné.

Otázka č. 9

Detailně vysvětlete, jaký je rozdíl mezi DMA a PIO přenosy. Jaké jsou jejich výhody a nevýhody? A můžeme provést DMA přenos z libovolného zařízení (řadiče), ze kterého potřebujeme tímto způsobem data dostat?

Otázka č. 10

Předpokládejte, že známe specifikaci varianty virtual machine vycházející z CLR (Common Language Runtime) = standardní VM .NETu. Naše VM je stroj s **harvardskou** a se **zásobníkovou registrovou** architekturou (pro jednoduchost předpokládejte, že všechny registry obsahují **64-bit floating-point** reálná čísla dle IEEE 754, velikost registrového zásobníku je neomezená). Strojový kód pro tuto VM budeme nazývat tzv. CIL kód (Common Intermediate Language). Víme, že v CIL kódu i v samotné virtual machine jsou všechna data uložena jako **little-endian**, a že instrukční sada VM obsahuje minimálně tyto instrukce (všechny mají jednobytový opcode následovaný případnými argumenty):

- `ldc.r8` – load constant, real 8 byte = load ve variantě immediate, kde argumentem instrukce je konstanta jako 64-bitové floating-point reálné číslo
- `ldsfld` – load static field = load z globální proměnné, jejíž adresa je argumentem instrukce
- `stsfld` – store static field = store do globální proměnné, jejíž adresa je argumentem instrukce
- `call` – volání procedury nebo funkce (argumenty se předávají zleva doprava na registrovém zásobníku a **odstraňuje je volaný**, návratová hodnota se předává na vrcholu registrového zásobníku)
- `ret` – návrat z podprogramu (bez explicitních argumentů)
- `add` – sčítání (bez explicitních argumentů)
- `sub` – odečítání (bez explicitních argumentů), pravý implicitní argument (tj. hodnota, která je odečítána) je ten, který je na vrcholu zásobníku
- `mul` – násobení (bez explicitních argumentů)
- `div` – dělení (bez explicitních argumentů) , pravý implicitní argument (tj. jmenovatel) je ten, který je na vrcholu zásobníku
- `neg` – unární operace negace (bez explicitních argumentů)
- `br` – nepodmíněný skok (branch)
- `ble` – podmíněný skok branch if less or equal to: instrukce odebere ze zásobníku dvě hodnoty, a pokud je druhá odebraná menší nebo rovná první odebrané, tak se provede skok na adresu, která je argumentem instrukce.

Napište v Pascalu bez použití inline assembleru kód procedury (i s deklarací), která by mohla být běžným překladačem přeložena do níže uvedeného kódu, který jsme ze spustitelného souboru obsahujícího CIL kód disassemblovali do assembleru CILu:

```
  ldsfld     [0x20600010]
  ldsfld     [0x20600000]
  sub
  ldsfld     [0x20600010]
  ldsfld     [0x20600000]
  sub
  mul
  ldsfld     [0x20600018]
  ldsfld     [0x20600008]
  sub
  ldsfld     [0x20600018]
  ldsfld     [0x20600008]
  sub
  mul
  add
  call       0x20581000
  ldc.r8     1.0
  ble        LABEL1
  ldsfld     [0x20600000]
  stsfld     [0x20600020]
  ldsfld     [0x20600008]
  stsfld     [0x20600028]
  br         LABEL2
LABEL1:
  ldsfld     [0x20600010]
  stsfld     [0x20600020]
  ldsfld     [0x20600018]
  stsfld     [0x20600028]
LABEL2:
  ret
```