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).

Otázka č. 1

Co nejefektivněji rozhodněte, která z následujících čísel jsou dělitelná 64. Stručně napište, jak jste postupovali.

(a) `$791e4df2b31a65c8` (b) `$5f8ae4f00000f000`
(c) `$2771c294493f58df` (d) `$82afdda1ff44ed7c`
(e) `$493d809c2c87ac00` (f) `$ea10c71b143ab0f4`
(g) `$0199a2aeaaed49dc` (h) `$000b513e24480b5a`
(j) `$08cff6315eb80f80` (k) `$31cbef005e1ea910`

Otázka č. 2

Předpokládejte, že v běžném programu v Pascalu voláme standardní proceduru `Write` s parametrem `42`, která způsobí, že po přeložení a spuštění aplikace pod OS Linux se hodnota `42` vypíše na obrazovku. Kde je procedura `Write(x : longword)` naimplementována? Jak asi vypadá její implementace a jak deklarace (argumenty/návratové hodnoty) všech jí případně volaných funkcí, které také ještě poběží v uživatelském režimu procesoru, pokud bychom je zapsali v Pascalu?

Společná část pro otázky označené X

Předpokládejte, že máme variantu CPU Intel 80386, který je **32-bitový little-endian** procesor s obecnou registrovou architekturou a s 32-bitovým logickým i fyzickým adresovým prostorem (logická adresa se přímo rovná adrese fyzické – tedy předpokládejte, že se **nevyužívá stránkování**).
Procesor má obecné registry EAX, EBX, ECX, EDX, ESI, EDI, EBP, 32-bitový příznakový registr EFLAGS s běžnými příznaky, registr ESP (stack pointer, ukazuje na poslední využitý byte, roste dolů), a registr EIP (instruction pointer). U obecných registrů existují také jejich 16-bitové varianty AX, BX, CX, DX, SI, DI, BP, které jsou pohledem na spodních 16-bitů původních registru, dále má CPU i 8-bitové varianty AL, BL, CL, DL, které jsou pohledem na nejnižších 8-bitů.
V instrukční sadě jsou **instrukce pro standardní bitové operace** a dále mimo jiné **i následující instrukce** (příznakový registr modifikují pouze aritmetické operace, ale instrukce přenosu dat nikoliv):

- `MOV reg,imm/[addr]` (load register)
- `MOV [addr],reg` (store register)
- `MOV reg0,reg1` (transfer from `reg1` to `reg0`)
- `ADD reg,imm/[addr]/reg` (add without carry)
- `SUB reg,imm/[addr]/reg` (subtract without carry)
- `IMUL EAX,EDX` (32-bit integer multiplication)
- `CMP reg,[addr]` (32-bit compare)
- `PUSH reg` (push)
- `POP reg` (pop)

Všechny uvedené instrukce i bitové operace se dvěma operandy mají vždy vlevo cílový a vpravo zdrojový operand, dále mají všechny 32-bitovou, 16-bitovou, i 8-bitovou variantu (zvolenou dle zdrojového, resp. cílového registru).
Dále má CPU ještě instrukce:

- `JMP addr` (direct jump), `JZ addr` (jump if zero)
- `CALL addr` (direct call), `CALL [addr]` (indirect call)
- `RET` (return from subroutine)

Obecně mohou mít instrukce tohoto CPU jednu z následujících variant operandů (povolené varianty viz definice konkrétní instrukce):

- 32-bit/16-bit/8-bit immediate: `imm`
- absolutní adresa: `[addr]`, kde `[addr]` může být jedno z:
  - `[imm]` adresa daná konstantou
  - `[reg +/- imm]` adresa daná součtem/rozdílem obsahu registru `reg` a konstanty `imm`
- libovolný registr: `reg`

Otázka č. 3 (X)

Napište v Pascalu bez použití inline assembleru kód procedury (i s deklarací), která by mohla být běžným překladačem přeložena do dále uvedeného kódu, který máme zapsaný v assembleru 80386 (víme, že procedura používá běžnou Cčkovou volací konvenci, tj. argumenty se předávají na volacím zásobníku zprava doleva, a odebírá je volající):

```
  push   ebp
  mov    ebp,esp
  sub    esp,0x8
  mov    edx,0x0
  mov    [ebp-0x8],edx
label1:
  mov    eax,[ebp-0x8]
  cmp    eax,[ebp+0xc]
  jz     label4
  mov    edx,0x0
  mov    [ebp-0x4],edx
label2:
  mov    eax,[ebp-0x4]
  cmp    eax,[ebp+0x8]
  jz     label3
  mov    edx,[ebp-0x8]
  mov    eax,[ebp+0x8]
  imul   eax,edx
  add    eax,[ebp-0x4]
  shl    eax,0x2
  add    eax,[ebp+0x10]
  mov    edx,0x0
  mov    [eax],edx
  mov    edx,[ebp-0x4]
  add    edx,0x1
  mov    [ebp-0x4],edx
  jmp    label2
label3:
  mov    edx,[ebp-0x8]
  add    edx,0x1
  mov    [ebp-0x8],edx
  jmp    label1
label4:
  mov    esp,ebp
  pop    ebp
  ret
```

Otázka č. 4 (X)

Předpokládejte, že chceme navrhnout novou procesorovou architekturu vycházející z Intel 80386. Nicméně stejné instrukce chceme zachovat pouze na úrovni assembleru, strojový kód pro nový procesor chceme navrhnout znovu a lépe. Navrhněte, jak by ve strojovém kódu měly vypadat všechny uvedené varianty instrukce `ADD` – uvažujte pouze práci s 32-bitovými registry a argumenty, u varianty s absolutní adresou uvažujte pouze podvariantu `[imm]`, rozlišení registrů proveďte různými variantami samotného opcode instrukce. Strojový kód můžete navrhnout tak, že různé varianty instrukce budou ve strojovém kódu různě dlouhé (snažte se je ale navrhnout co nejkratší). Pro jednoduchost se neohlížejte na ostatní instrukce. Pro každý byte vámi navrženého instrukčního kódu popište jeho význam.

Otázka č. 5

Vysvětlete, co to je tzv. *stack overflow* a co tzv. *stack underflow*. Může k daným situacím dojít při volání běžné funkce napsané v Pascalu? Detailně vysvětlete proč.

Otázka č. 6

Předpokládejte, že implementujeme RTL jazyka Pascal a naším úkolem je v Pascalu naprogramovat standardní funkci `IntToHex`, která má předané číslo převést do šestnáctkové soustavy (stejně jako fce ve Free Pascalu, jak ji známě z přednášek). Nicméně pro jednoduchost předpokládejte, že má funkce pouze jeden argument, a že výsledek má vždy tolik platných číslic kolik odpovídá maximálního rozsahu zvoleného vstupního typu (tedy číslo je zleva dorovnané nulami). Napište implementaci takové funkce včetně deklarace. Očekávejte, že typ `char` je 1 bytový, a znak v něm uložený je v 8-bitovém rozšíření kódování ASCII.

Otázka č. 7

Předpokládejte nějakou běžnou implementaci jednosměrně vázaného seznamu. Pokud je jeden seznam sdílený 2 vlákny v jednom programu běžícím na nějakém typickém operačním systému s preemptivním přepínám vláken (např. OS Windows 10), a pokud vlákna nepoužívají žádná synchronizační primitiva pro ochranu seznamu, může dojít k nějakému naplánování vláken, že při vládání nebo odebírání položek ze seznamu dojde k race condition, která způsobí, že bude seznam v nekonzistentním stavu vzhledem k pohledu alespoň jednoho z vláken?
Pokud ano, tak napište konkrétní proložení příkazů vláken, kdy by k takové situaci mohlo dojít a vznikající problém vysvětlete. Pokud ne, tak detailně vysvětlete, proč k race condition nemůže dojít.

Otázka č. 8

Předpokládejte, že chceme reálné číslo `1043,75` uložit do Pascalové proměnné ve standardním formátu single. Typ `single` je 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.
Nyní program obsahující takovou proměnnou spustíme na počítači s 32-bitovým little-endian CPU, do proměnné uložíme uvedenou hodnotu `1043,75`, a zjistili jsme, že je proměnná uložená na adrese `0x5C06BB00`. Napište v šestnáctkové soustavě hodnotu každého bytu paměti, ve kterém bude uložená nějaká část proměnné.

Otázka č. 9

Naprogramujte v Pascalu program, který jako 1. argument na příkazové řádce – dostupný jako `string` výsledek běžné Pascal funkce `ParamStr(1)` – dostane jméno souboru s binárním obrazem disku, o které víme, že používá standardní formát master boot sektoru (recordu) – viz dále – a dále víme, že je rozdělený na několik oddílů. Úkolem programu je pro každý oddíl označený jako aktivní/bootovatelný vypsat C/H/S adresu jeho prvního a posledního sektoru – tedy pokud by např. např. byl bootovatelný pouze 2. oddíl (mezi 1/2/3 a 10/4/5) a 4. oddíl (mezi 20/6/7 a 40/8/9), tak má váš program vypsat následující text:

```
2: 1/2/3 - 10/4/5
4: 20/6/7 - 40/8/9
```

Při tvorbě programu využijte následující specifikaci, kterou jsme našli na Wikipedii:
A master boot record (MBR) is a special type of boot sector at the very beginning of partitioned computer mass storage devices like fixed disks or removable drives intended for use with IBM PC-compatible systems and beyond. By convention, there are exactly four primary partition table entries in the MBR partition table scheme – structure of a classical generic MBR:

Offset	*|B|*	*Description*
0x000	*446*	*Bootstrap code area*
0x1BE	*16*	*Partition entry number 1*
0x1CE	*16*	*Partition entry number 2*
0x1DE	*16*	*Partition entry number 3*
0x1EE	*16*	*Partition entry number 4*
0x1FE	*1*	*0x55 first byte of MBR signature*
0x1FF	*1*	*0xAA second byte of MBR signature*

Total size: 446 + 4×16 + 2 = 512
Layout of one 16-byte partition entry (all multi-byte fields are little-endian):

Offset	*|B|*	*Description*
0x00	*1*	*Status of physical drive (bit 7 is set for active or bootable partition)*
0x01	*1*	*Bits 7-0: 8-bit **head** number of CHS address of **first** absolute sector in partition.*
0x02	*1*	*Bits 7-6: 2 most significant bits of 10-bit **cylinder** number of CHS address of **first** absolute sector in partition.* *Bits 5-0: 6-bit **sector** number of CHS address of **first** absolute sector in partition.*
0x03	*1*	*Bits 7-0: 8 least significant bits of 10-bit **cylinder** number of CHS address of **first** absolute sector in partition.*
0x04	*1*	*Partition type*
0x05	*1*	*Bits 7-0: 8-bit **head** number of CHS address of **last** absolute sector in partition.*
0x06	*1*	*Bits 7-6: 2 most significant bits of 10-bit **cylinder** number of CHS address of **last** absolute sector in partition.* *Bits 5-0: 6-bit **sector** number of CHS address of **last** absolute sector in partition.*
0x07	*1*	*Bits 7-0: 8 least significant bits of 10-bit **cylinder** number of CHS address of **last** absolute sector in partition.*
0x08	*4*	*LBA of first absolute sector in the partition*
0x0C	*4*	*Number of sectors in partition*

Zdroj: [http://en.wikipedia.org/wiki/Master_boot_record]

Otázka č. 10

Detailně vysvětlete, k čemu na I^2C sběrnici slouží tzv. *repeated start*, a zda by bylo možné se bez něj obejít. Jakým způsobem se *repeated start* rozpozná od běžného bitu? A proč podobný koncept neexistuje na RS-232 lince?