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).

```
00000000: 00 19 50 00 61 00 76 00|65 00 6C 00 20 00 4A 00  | ..P.a.v.e.l. .J.
00000010: 65 00 7E 01 65 00 6B 00|21 00 00 00 CD 90 97 20  | e.~.e.k.!...Í.-
00000020: BC B3 08 4D A3 B8 C6 C1|19 FF F3 C0 00 04 20 01  | Ľł.MŁ¸ĆÁ·Ŕó... .
00000030: 01 08 03 20 00 01 05 20|01 01 11 11 04 20 01 01  | ... ... ..... ..
```

Společná část pro otázky označené X

Předpokládejte, že jsme si pomocí hex vieweru zobrazili obsah začátku souboru v nějakém datovém formátu – viz výpis výše v záhlaví tohoto zadání.

Otázka č. 1 (X)

O formátu souboru víme, že od 2. bytu (počítáno od nuly) obsahuje textový řetězec, ale nevíme, v jakém je řetězec formátu. Dle uvedeného příkladu rozhodněte a své rozhodnutí detailně zdůvodněte, jaký má asi textový řetězec formát a v jakém kódování je nejspíše uložen. Variantu kódování určete co nejpřesněji.

Otázka č. 2 (X)

Víme, že v zobrazeném souboru je od 40. bytu (zapsáno v desítkové soustavě, a počítáno od nuly) uloženo 16-bitové znaménkové číslo ve dvojkovém doplňku. Číslo je uložené jako **little-endian**. Napište v desítkové soustavě hodnotu tohoto uloženého čísla včetně jeho znaménka.

Otázka č. 3 (X)

Předpokládejte, že máme k jednočipovému počítači přes I^2C sběrnici připojenu 64-Kbit (8K × 8) F-RAM paměť (což je varianta non-volatile paměti, pro účel této otázky ekvivalentní s EEPROM) *Cypress FM24C64B* (datasheet je přílohou tohoto zadání – pro tuto otázku jsou důležité hlavně části „*Acknowledge/No-acknowledge*“, „*Slave Device Address*“, „*Addressing Overview*“, „*Write Operation*“). Paměť budeme mít zapojenou tak, že piny `A0` až `A2` budou připojené na zem. Nyní během jedné zápisové transakce zapíšeme od adresy 148 do uvedené F-RAM paměti první dva byty výše uvedeného souboru. Zapište, jaká posloupnost bitů bude v průběhu celé takové transakce vyslána po vodiči SDA I^2C sběrnice (pro jednotlivé skupiny bitů označte jejich význam, nejvíce vlevo pište bit, který bude poslaný jako první).

Otázka č. 4

Předpokládejte, že na základní desce jednoduchého jednoúčelového počítače chceme navrhnout jednoduchou paralelní paměťovou sběrnici pro připojení jednoho SRAM paměťového modulu s 16-bitovým slovem o celkové kapacitě 32 kB. Víme, že další paměťové moduly připojovat nebudeme, a ani modul s jinou než uvedenou kapacitou. Dále předpokládejte, že pro počítač později vybereme vhodnou variantu 16-bit CPU, který bude umožňovat přímé připojení SRAM modulu přímo pomocí navržené paměťové sběrnice. Jaké všechny signály/vodiče taková sběrnice bude minimálně mít? U každého signálu vysvětlete jeho význam.

Otázka č. 5

Předpokládejte, že máme v systému s preemptivním přepínáním vláken k dispozici standardní implementaci zámků/mutexů (`TLock`) a operací jejich zamykání (`Lock`) a odemykání (`Unlock`):

```
type PLock = ^TLock;
     TLock = record
       ...
     end;
procedure Lock(lock : PLock); begin ... end;
procedure Unlock(lock : PLock); begin ... end;
```

Dále v systému spustíme současně dvě vlákna (1. s hlavní procedurou `ThreadX`, 2. s hlavní procedurou `ThreadY`) s následující implementací:

```
var
  lock1, lock2, lock3 : PLock;

procedure ThreadX;
begin
  Lock(lock1);
  ...
  Lock(lock2);
  ...
  Unlock(lock2);
  Unlock(lock1);
end;

procedure ThreadY;
begin
  Lock(lock2);
  ...
  Lock(lock1);
  ...
  Lock(lock3);
  ...
  Unlock(lock3);
  Unlock(lock1);
  Unlock(lock2);
end;
```

Může dojít při běhu těchto vláken k deadlocku? Zdůvodněte tak, že pro každou ze 4 Coffmanových podmínek ukažte, zda v dané situaci pro uvedené zdroje = zámky platí, či nikoliv.

Otázka č. 6

Předpokládejte, že máme následující globální 2 proměnné:

```
x : longword
y : longword
```

O proměnné `x` víme, že v ní máme uloženu hodnotu α z rozsahu 0-5789. Proměnná `y` reprezentuje data, která budeme ukládat do registru řadiče, kterému chceme předat hodnotu α. Dle komunikačního protokolu řadiče vyžadujeme, že nejnižších 10 bitů hodnoty α má být uložených v bitech 3 až 12 (LSb je bit 0) proměnné `y`, zbylé 3 nejvyšší bity hodnoty α v bitech 24 až 26 proměnné `y`. Zapište v Pascalu posloupnost aritmetických a bitových operací, která z proměnné `x` zkonstruuje správný obsah proměnné `y` (hodnota jiných než uvedených bitů proměnné `y` by měla být zachována). Typ `longword` je 32-bitové bezznaménkové číslo.

Společná část pro otázky označené Y

Předpokládejte níže popsaný CPU vycházející z architektury procesorů Intel 80386 – je to **32-bitový little-endian** CPU s obecnou registrovou architekturou, s podporou stránkování a s 32-bitovým virtuálním i fyzickým adresovým prostorem. Procesor má obecné registry `EAX`, `EBX`, `ECX`, `EDX`, `DSI`, `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). V instrukční sadě jsou mimo jiné **i následující instrukce** (příznakový registr modifikují pouze aritmetické operace, ale instrukce přenosu dat nikoliv):

- `MOV reg,DWORD PTR imm32/[addr]` (load register)
- `MOV DWORD PTR [addr],reg` (store register)
- `MOV reg0,reg1` (transfer from `reg1` to `reg0`)
- `SUB reg,imm32/[addr]/reg` (subtract without carry)
- `PUSH imm16/imm32/[addr]/reg`, `POP [addr]/reg`
- `JMP addr` (direct jump), `JNE addr` (jump if not equal)
- `JB addr` (jump if below, tj. když `carry = 1`)
- `CMP reg,DWORD PTR [addr]` (32-bit compare)
- `INT imm8` (software interrupt/syscall)
- `RET` (return from subroutine)

Všechny výše uvedené instrukce se dvěma operandy mají vždy **vlevo cílový** a **vpravo zdrojový** operand. Instrukce mohu mít jednu z následujících variant operandů (povolené varianty viz definice konkrétní instrukce):

- 8-bitový/32-bitový immediate `imm8`/`imm32`
- absolutní adresa `[addr]`, kde `[addr]` může být:
  - `[reg +/- imm]` adresa daná součtem/rozdílem obsahu registru `reg` a konstanty `imm`
- libovolný registr `reg`

Otázka č. 7 (Y)

Napište v Pascalu bez použití inline assembleru kód funkce (i s deklarací), která by mohla být běžným překladačem přeložena do níže uvedeného kódu v assembleru 80386 (předpokládejte, že funkce používá běžnou Cčkovou volací konvenci, tj. argumenty se předávají na volacím zásobníku zprava doleva, a **odebírá je volající**, návratová hodnota je uložená v registru `EAX`):

```
  push  ebp
  mov   ebp,esp
  sub   esp,0x4
  mov   eax,DWORD PTR [ebp+0x8]
  cmp   eax,DWORD PTR [ebp+0xc]
  jb    LABEL3
  mov   eax,DWORD PTR [ebp+0x8]
  cmp   eax,DWORD PTR [ebp+0xc]
  jne   LABEL1
  mov   DWORD PTR [ebp-0x4],0x0
  jmp   LABEL2
LABEL1:
  mov   DWORD PTR [ebp-0x4],0x1
LABEL2:
  jmp   LABEL4
LABEL3:
  mov   DWORD PTR [ebp-0x4],0xffffffff
LABEL4:
  mov   eax,DWORD PTR [ebp-0x4]
  mov   esp,ebp
  pop   ebp
  ret
```

Společná část pro otázky označené Z

Předpokládejte, že implementujeme část standardní runtime knihovny naší nové implementace jazyka Pascal, která se stará o správu Pascalové haldy. Součástí naší implementace je již následující kód:

```
type PByte = ^byte; PBlock = ^TBlock;
     TBlock = record
       Size : longword;
       Next : PBlock;
     end;
var firstFree : PBlock; lastFree : PBlock;
function KeAllocMem (pageCount : longword)
  : PByte; cdecl; begin ... end;
function AddFreeBlock : boolean; begin ... end;
```

Halda je uspořádaná tak, že každý volný blok paměti na haldě začíná N bytovou hlavičkou ve formě záznamu `TBlock`, a hned za posledním bytem záznamu `TBlock` následuje M bytů samotné volné paměti. Pro každý volný blok je v položce `Size` uloženo číslo M (tj. velikost volné paměti bloku bez započítání velikosti hlavičky bloku – tj. celkem blok zabírá N+M bytů). Všechny volné bloky jsou pomocí položky `Next` v hlavičce bloku uspořádány do jednosměrně vázaného seznamu, kde na první blok ukazuje proměnná `firstFree`, na poslední blok ukazuje proměnná `lastFree`. Můžete předpokládat, že **na haldě je vždy alespoň 1 volný blok**. Dále máme připravenu funkci `KeAllocMem`, která volá API funkci `AllocMem` jádra cílového operačního systému, která alokuje kontinuální blok `pageCount` stránek (**velikost stránky je 4 kB**), a vrací bázovou adresu takového bloku, nebo **`nil`** pokud již OS nemá k dispozici volné stránky.

Otázka č. 8 (Y, Z)

Předpokládejte, že cílový OS má následující ABI – pro vstup do jádra se používá softwarové přerušení `$80`, volání funkce `AllocMem` je identifikováno konstantou `5` v registru `EAX`, 1. parametr funkce se předává v registru `EBX`, návratová hodnota v registru `EAX`. Dále víme, že pro funkci `KeAllocMem` s Cčkovou volací konvencí (viz otázka č. 7) překladač Pascalu generuje prázdný prolog funkce a v epilogu nemodifikuje obsah registru `EAX`. Doplňte kód funkce `KeAllocMem` tak, aby volal funkci `AllocMem` jádra OS.

Otázka č. 9 (Y, Z)

Napište v Pascalu implementaci funkce `AddFreeBlock` tak, aby si od OS vyžádala 1 stránku volné paměti. Pokud nová stránka navazuje na poslední volný blok na haldě, tak se má tento blok (informace v jeho hlavičce) prodloužit o velikost stránky. Pokud na poslední blok nenavazuje, tak založte nový blok volné paměti a vložte ho na konec seznamu volných bloků (tj. na začátku nově naalokované stránky správně vyplňte záznam `TBlock`). Funkce má vrátit `false`, pokud není k dispozici volná stránka. V kódu **nesmíte** použít funkci `New`, jelikož ta bude váš kód využívat.

Otázka č. 10 (Y, Z)

Mějme typický překladač Pascalu a globální proměnnou `x` typu **`array`**`[0..3]` **`of`** `TBlock`. Jaká bude nejspíš celková velikost paměti zabraná proměnnou `x`, je-li `longword` 32-bitové bezznaménkové číslo? Na jakém offsetu od její bázové adresy bude ležet položka `x[2].Size` a na jakém položka `x[2].Next`? Detailně vysvětlete proč.