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

Vysvětlete to je *analogový* a co *digitální* přenos dat, a dále vysvětlete, jaké jsou mezi nimi rozdíly. Jaké jsou výhody a nevýhody obou přístupů?

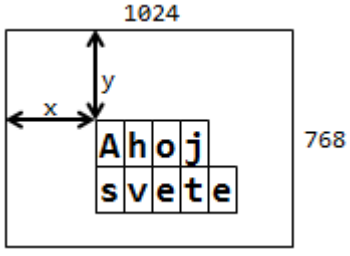
Otázka č. 2

V kontextu OS běžícího na **32-bit little-endian** CPU **bez podpory stránkování** (a kde se logické adresy přímo rovnají fyzickým) naimplementujte v Pascalu následující proceduru ovladače grafické karty:

```
type
  PByte = ^byte;
  TCharBitmap = array[0..15] of byte;
  PFont = ^TFont;
  TFont = array[0..255] of TCharBitmap;
procedure Print(
  x : longword; y : longword;
  str : PByte; font : PFont);
```

která má na zadané souřadnice na obrazovce (parametry `x` a `y` reprezentují souřadnice levého horního rohu obdélníku opsaného výslednému textu) zobrazit předaný null-terminated řetězec `str` v 8-bitovém rozšíření kódování ASCII jako černý text na bílém pozadí (černá má všechny barevné složky rovné `0`, bílá je má všechny rovné maximální hodnotě). Parametr `font` ukazuje na pole, kde každá jeho položka reprezentuje bitmapu pevné velikosti 8x16 pixelů, která je vizualizací znaku s kódem odpovídajícímu indexu v poli `font`. Pixely mají v této bitmapě 1-bitvou hloubku, kde MSb reprezentuje nejlevější pixel řádku, LSb reprezentuje nejpravější pixel řádku. Obrázky jednotlivých znaků se mají na obrazovce zobrazit postupně zleva doprava jeden vedle druhého (bez mezer). Počítejte s tím, že předané souřadnice a text jsou vhodně zvolené tak, že se zobrazení textu vejde na jeden řádek na obrazovce – pokud ale text obsahuje znak konce řádku (vyberte nějakou běžnou reprenzetaci), tak se mají další znaky zobrazovat opět od sloupce `x`, ale o 16 pixelů níže než předcházející text. Framebuffer grafické karty začíná na fyzické adrese `$000A0000`, používá standardní 24-bitový formát pixelů, a grafická karta je nakonfigurována pro rozlišení 1024x768 pixelů.

Chování vaší funkce bude tedy např. pro předaný text:
`Ahoj`$_{newline}$`svete`
v kontextu popsané grafické karty následující:



Otázka č. 3

Popište a na příkladu vysvětlete další možnosti ukládání textových řetězců jako alternativy k null-terminated řetězcům. Porovnejte výhody a nevýhody jednotlivých uvedených přístupů.

Otázka č. 4

Bylo by potřeba něco ve vaší proceduře z **otázky 2** změnit, nebo v OS něco nějak speciálně nakonfigurovat, pokud by měla být napsaná pro OS běžícím na CPU **s podporou stránkování**, pokud je stránkování opravdu zapnuté a plně využívané jako v běžném moderním OS jako Windows nebo Linux? Detailně vysvětlete, proč by žádné úpravy nebyly potřeba, nebo detailně vysvětlete, jaké úpravy provedete.

Otázka č. 5

Předpokládejte, že navrhujeme systém, kde k nějakému typickému 32-bitovému CPU (s 32-bitovým adresovým prostorem) chceme připojit 2 paměťové moduly typu SRAM, kde každý z nich máme ve variantě `4096x16`. Víme, že CPU používá skupinu vodičů pro přenos adresy a další nezávislou skupinu vodičů pro přenos dat. Jak by za této situace vypadalo zapojení celého paměťového subsystému počítače? Nakreslete obrázek hlavních komponent, které bude obsahovat, a vodičů, které mezi nimi povedou. Pokud to dává smysl, tak vodiče s podobnou funkcí můžete v obrázku seskupit a napsat k nim vhodný komentář. Pokud budete do obrázku kreslit nějaké další komponenty než samotné CPU a SRAM paměťové čipy, tak u takových komponent popište jejich chování.

Otázka č. 6

Navrhněte HCI řadiče pevných disků, který má být připojitelný na paralelní 32-bitovou systémovou sběrnici PCI. Řadič má samotná data sektorů přenášet do paměti a z paměti výhradně sám pomocí tzv. DMA bus masteringu. Detailně popište strukturu a funkci všech registrů, které takový řadič bude mít – pro uvedené registry zvolte nějakou vhodnou bázovou adresu v adresovém prostoru, když víme, že firmware cílových počítačů bude uložený v 256 kB paměti EEPROM a maximální velikost instalované paměti DRAM bude 1 GB. Dále pro řadič popište veškeré další informace, které o řadiči bude SW s ním komunikující potřebovat vědět.

Otázka č. 7

V kontextu moderního OS s podporou pro vícevláknové zpracování napište v Pascalu implementaci funkce `ReadSector` ovladače pevných disků:

```
  procedure ReadSector(
    sectorNumber : longword; buffer : pointer);
```

když víte, že je taková funkce vždy volána v kontextu nějakého aplikačního vlákna. Předpokládejte, že řadič pevných disků používá HCI, které jste si navrhli v **otázce 6**. Můžete očekávat, že od jádra (resp. jeho HAL vrstvy) máte k dispozici běžné procedury a funkce, které můžete ve vašem kódu využít (volat) – pro každou takovou proceduru/funkci jádra, kterou ve svém kódu použijete, tak napište její deklaraci (jaké bude mít parametry/návratové hodnoty) a stručně popište její chování. Dále předpokládejte, že jsou vám přístupné běžné datové struktury, které by jádro takového OS využívalo.

Společná část pro otázky označené X

Předpokládejte, že máme počítač, kde jako systémová sběrnice slouží standardní varianta paralelní 32-bitové PCI sběrnice s multiplexovanými adresovými a datovými vodiči. V počítači máme nainstalovánu variantu CPU Intel 80386 běžící v tzv. chráněného režimu (protected mode 32). V tomto režimu se CPU chová jako **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)
- `CMP reg,[addr]` (32-bit compare)
- `PUSH SIZE PTR imm/[addr]/reg` (push)
- `POP SIZE PTR [addr]/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). U instrukcí `PUSH` a `POP` se místo `SIZE` dosadí slovo `DWORD` nebo `WORD` nebo `BYTE` pro výběr jedné z uvedených variant instrukce. Dále má CPU ještě instrukce:

- `JMP addr` (direct jump), `JA addr` (jump if above >)
- `CALL addr` (direct call), `CALL [addr]` (indirect call)
- `RET` (return from subroutine)
- `INT n` (software interrupt)
- `IRET` (return from interrupt)

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`

Tabulka vektorů přerušení leží od adresy `0x00000000`, tabulka obsahuje 256 vektorů pro přerušení `0` až `255`, každý vektor je právě jedna logická adresa.

Otázka č. 8 (X)

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

```
  SUB ESP, 4
  MOV EAX, 0
  MOV [ESP], EAX
label1:
  MOV EAX, [ESP]
  CMP EAX, [ESP+08h]
  JA  label2
  MOV ESI, [ESP+0Ch]
  MOV EAX, [ESI]
  MOV EDI, [ESP+10h]
  MOV EBX, [EDI]
  ADD EAX, EBX
  SHR EAX, 1
  MOV [ESI], EAX
  ADD ESI, 4
  MOV [ESP+0Ch], ESI
  ADD EDI, 4
  MOV [ESP+10h], EDI
  MOV EAX, [ESP]
  ADD EAX, 1
  MOV [ESP], EAX
  JMP label1
label2:
  MOV EAX, [ESP+0Ch]
  ADD ESP, 4
  RET
```

Otázka č. 9 (X)

Jak by na PCI mohl vypadat jeden časový diagram ilustrující všechny přenosy za následující situace: Ve stejném okamžiku se CPU rozhodlo zapsat hodnotu `0x00AA6708` na adresu `0x20580000`, a řadič pevného disku data `7E 00 00 60 20 E8 D8 FF FF FF FF A3` od adresy `0x00401964`, a víme, žádné další přenosy na sběrnici neprobíhají. Pokud více vodičů seskupíte do jednoho řádku, tak napište v šestnáctkové soustavě jejich souhrnnou hodnotu. Ke všem použitím vodičům napište stručný komentář jejich významu.

Otázka č. 10 (X)

Předpokládejte, že programujeme část jednoduchého operačního systému bez podpory vícevláknového zpracování – naším úkolem je naprogramovat níže uvedenou proceduru (a máme k dispozici lib. glob. prom.):

```
procedure ReadSector(n : longword; buffer : pointer);
```

která z pevného disku číslo `0` přečte sektor číslo `n` na adresu `buffer`. Napište ve Free Pascalu tuto proceduru s využitím API procedury firmware našeho počítače (deklarace viz níže) – pozor, přestože **náš kód** poběží **v chráněném 32-bitovém** režimu CPU, tak **firmwarová API** funkce přepne CPU to tzv. reálného režimu CPU, kdy se 80386 chová jako původní CPU Intel 8086: je to **16-bitový little-endian** CPU s podporou 20-bitového fyzického adresového prostoru. Logické adresy jsou 32-bitové, kde spodních 16 bitů logické adresy je tzv. `offset`, a horních 16 bitů je tzv. `segment` – přepočet z logické na fyzickou adresu je následující: `fyzická = segment * 16 + offset` (tedy 1 fyzická adresa je ekvivalentně reprezentována mnoha různými logickými adresami). U našeho CPU 80386 je v reálném režimu spodních 20 bitů fyzické adresy získáno stejně jako u 8086, horních 12 bitů je vždy nulových. Víme, že adresa předaná v `buffer` má vždy horních 12 bitů nulových.

Čtení sektoru zpřístupňuje firmware procedura číslo 42h, volaná obsluhou softwarového přerušení číslo 13h – parametry má předávané v registrech:

- `AL` = 42h, `BL` = číslo zdrojového pevného disku
- `CX:SI` = dvojice registrů obsahuje adresu cílového bufferu ve formě segment:offset
- `DX` = horních 16 bitů, `DI` = dolních 16 bitů čísla sektoru