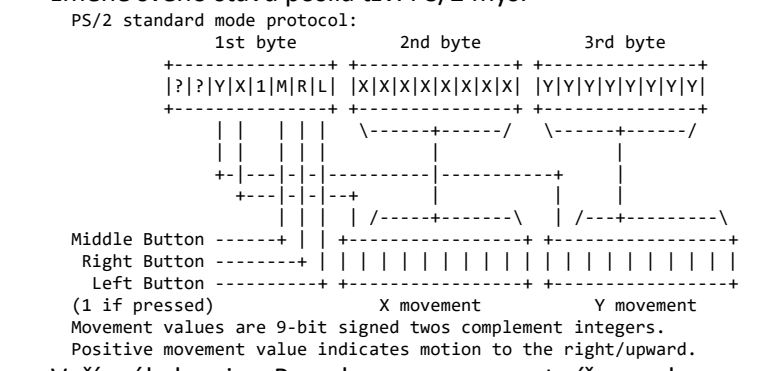
Odpovědi pište na zvláštní odpovědní list s vaším jménem a fotografií. Pokud budete odevzdávat více než jeden list s řešením, tak se na 2. a další listy nezapomeňte podepsat. Do zápatí všech listů vždy napište i/N (kde i je číslo listu, N je celkový počet odevzdaných listů).

Otázka č. 1

Našli jsme následující specifikaci paketu, který při každé změně svého stavu posílá tzv. PS/2 myš:



Vaším úkolem je v Pascalu naprogramovat níže uvedenou proceduru:

```
type PByte = ^byte;
procedure DisplayMouseMove(packet : PByte);
```

která v parametru `packet` získá ukazatel na 3 bytový paket přijatý od PS/2 myši, a jejímž úkolem je vypsat změny polohy myši v každé z os jako nezáporné číslo s popisem směru této změny – tedy např. pokud myší vhodnou rychlostí pohneme doprava, tak má zobrazit:

```
10 RIGHT 0 UP
```

Pokud vhodnou rychlostí pohneme doleva nahoru, zobrazí:

```
15 LEFT 5 UP
```

Pokud vhodnou rychlostí pohneme doprava dolů, zobrazí:

```
7 RIGHT 8 DOWN
```

Otázka č. 2

Předpokládejte nějakou typickou implementaci zámků v moderním operačním systému. Ukažte příklad situace, kdy by za použití takových zámků mohlo dojít k tzv. *deadlocku*. Pokud máme nějaký program používající takové zámky, dalo by se principiálně nějakým způsobem ověřit, že je napsaný tak, že v něm kvůli zámkům nikdy deadlocku dojít nemůže? Pokud ano, tak vysvětlete jak.

Otázka č. 3

Předpokládejte, že máme k dispozici následující proceduru, která je součástí RTL nějakého programovacího jazyka (níže uvedená deklarace je pro srozumitelnost ale zapsána v Pascalu):

```
type PWord = ^word; procedure Write(str : PWord);
```

Víme, že procedura text předaný ve formě null-terminated UTF-16 little-endian řetězce vypisuje na standardní výstup a chová se běžným způsobem. Pokud bychom **jedním voláním** této procedury chtěli na výstup vypsat **celý** následující 3 řádkový text:

```
I
LOVE
YOU
```

jak by v paměti vypadal vhodný řetězec, pokud víme, že je uložen od adresy `$000015C0`? Zapište v šestnáctkové soustavě hodnotu každého bytu, který takový řetězec v paměti zabírá. Pokud vaše řešení závisí na cílovém operačním systému, tak si nějaký vhodný zvolte, a vysvětlete, jak jeho volba ovlivňuje vaše řešení. Poznámka: znak `A` má v kódování ASCII přiřazený 7-bitový kód `0x41`.

Otázka č. 4

Procesor 6502 je **8-bitový little endian** mikroprocesor s akumulátorovou architekturou a s **16-bitovým paměťovým adresovým prostorem**. Kromě registru akumulátoru (v assembleru značený `A`) má procesor ještě dva pomocné registry `X` a `Y`. Procesor má HW podporu pro implementaci volacího zásobníku – volací zásobník musí ležet na adresách mezi `$100` a `$1FF`. Zásobník roste dolů, offset **prvního volného místa** na zásobníku je uložen v **8-bitovém** registru `S` (pozor: ve skutečnosti první volné místo na zásobníku leží na adrese `$100 + S`). Procesor 6502 má následující instrukční sadu (∀ instrukce má 1 byte opcode): `LDA arg` (Load Accumulator), `STA arg` (Store Accumulator), `PHA` (Push Accumulator) , `PLA` (Pull Accumulator = ekvivalent operace POP běžné na jiných procesorech), `NOP` (No Operation), `SEC` (Set Carry), `CLC` (Clear Carry), `ADC arg` (Add with Carry), `AND arg` (bitový and), `ORA arg` (bitový or), `EOR arg` (bitový xor), `SBC arg` (Subtract from A with Borrow), `INX` (INrement X – zvětší obsah registru X o jedna), `DEC addr` (DECrement – zmenší 8-bit cílovou paměťovou lokaci argumentu o jedna), `JSR addr` (Jump to SubRoutine – ekvivalent u jiných procesorů běžné instrukce CALL), `RTS` (Return from SubRoutine – ekvivalent RET), `JMP addr` (JuMP – nepodmíněný skok), `CMP addr` (CoMPare accumulator against argument), `BEQ addr` (Branch if EQual) , `BNE addr` (Branch if Not Equal) , `BLT addr` (Branch if Less Than, tj. skok když po compare A bylo ostře menší <), a 6 instrukcí pro kopírování hodnoty (Transfer) mezi registry (2. písmeno jména instrukce = zdrojový registr, 3. písmeno = cílový registr): `TXA`, `TAX`, `TYA`, `TAY`, `TSX`, `TXS`. Argument instrukcí (pokud ho mají) může být jedna z následujících variant: `#číslo` (= hodnota immediate), `číslo` (= adresa v paměti), `číslo,X` (= přímá adresa v paměti získaná výpočtem – výsledná adresa v paměti se získá jako *číslo* + X, tj. výsledek součtu hodnoty *číslo* a hodnoty uložené v registru X), podobně `číslo,Y` (= přímá adresa v paměti, tj. adresa *číslo* + 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 6502 (předpokládejte, že funkce používá běžnou Cčkovou volací konvenci, tj. argumenty se předávají na volacím zásobníku zprava doleva, a odebírá je volající, návratová hodnota v registru akumulátoru `A`):

```
    TSX
    LDA $103,X
    CLC
    ADC $104,X
    BNE label1
    LDA $103,X
    CMP $104,X
    BLT label4
    BEQ label4
 label1:
    LDA $105,X
    EOR #$FF
    AND $106,X
    BEQ label2
    LDA #$00
    JMP label3
label2:
    LDA $103,X
    ORA $105,X
label3:
    JMP label5
label4:
    LDA #$00
label5:
    RTS
```

Otázka č. 5

Předpokládejte, že navrhujeme systém, kde k typickému 16-bitovému CPU s 24-bitovým fyzickým adresovým prostorem chceme připojit 4 paměťové moduly typu SRAM, kde každý z nich máme ve variantě `1024x8`. 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.

Společná část pro otázky označené X

Předpokládejte počítač s 64-bitovým procesorem Intel Core, který podporuje běh moderních operačních systémů jako Windows nebo Linux. CPU je připojený na 32-bitovou variantu paralelní systémové sběrnice PCI se sdílenými adresovými a datovými vodiči. Na systémové sběrnici je mimo jiné připojen řadič pevných disků podporující DMA bus mastering, který vlastně slouží jako HBA sériové point-to-point sběrnice SATA, ke které máme připojený `4 TiB` pevný disk, který ještě využívá tradiční `512` bytové sektory.

Otázka č. 6 (X)

Víme, že uvedený procesor má ve své instrukční sadě kromě instrukce `RET` (return from subroutine) i speciální instrukci `IRET` (interrupt return). Vysvětlete, jaké hlavní kroky asi uvedený CPU provádí při vykonávání instrukce `IRET`. Dále vysvětlete, zda je taková speciální instrukce `IRET` opravdu nutná, a zda bychom si ve všech situacích nevystačili jen s instrukcí `RET`.

Otázka č. 7 (X)

Víme, že součástí HCI řadiče jsou je i W/O registr `SectorNumber`, jehož obsah určuje číslo sektoru, který má řadič z disku číst nebo na disk zapisovat, a W/O registr `TargetAddress`, který obsahuje fyzickou adresu v paměťovém adresovém prostoru sběrnice PCI, která určuje bázovou adresu bufferu v paměti, kde bude začínat prováděný DMA přenos. Zbytek HCI řadiče na navržený běžným způsobem.

Pokud má řadič plně podporovat uvedený disk a cílový systém, tak rozhodněte, jaká musí být minimální velikost registrů `SectorNumber` a `TargetAddress` **počítáno v celých bytech**. Vaše rozhodnutí detailně zdůvodněte.

Pokud řadič započne nějaký DMA přenos na základě požadavku ovladače tohoto řadiče, jak se takový ovladač pro nějaký typický OS dozví o dokončení DMA přenosu?

Otázka č. 8 (X)

Předpokládejte, že pevný disk uvedený v **části X** je rozdělen na 3 diskové oddíly: první oddíl zabírá sektory `$10000` až `$7FFFFF`, je zformátovaný souborovým systémem FAT32, a je na něm nainstalovaný operační systém A; druhý oddíl zabírá sektory `$800000` až `$FFFFFF`, je zformátovaný též souborovým systémem FAT 32, a je na něm nainstalovaný operační systém B; poslední třetí oddíl zabírá sektory `$1000000` až `$1FFFFFFF`, je naformátovaný souborovým systémem NTFS, a slouží pouze pro ukládání dat a není bootovatelný.

Po zapnutí počítače se nám zobrazí informace o celkové kapacitě paměti počítače, následované textem „`Press F1 to enter computer setup`“ (a víme, že stiskem klávesy F1 bychom se dostali to „konfiguračního programu počítače“). Pokud konfigurační program nespustíme, zobrazí se nám na obrazovce výběr, zda chceme nabootovat do operačního systému A nebo do operačního systému B. Zvolili jsme operační systém B, a na obrazovce se zobrazilo „`Loading B kernel ...`“, a po další chvíli se zobrazila příkazová řádka operačního systému, kde můžeme zadávat různé příkazy a spouštět libovolné programy. Popište, jaký kód vypisuje všechny výše uvedené texty, a jaký kód čeká na příkazy uživatele na konci bootování. Kde se každý takový kód vezme (kde je uložený při vypnutém počítači), a jak CPU ví, že ho má zpracovávat? Vše detailně vysvětlete.

Otázka č. 9

Vysvětlete v kontextu sběrnice I^2C, co to je tzv. *burst* přenos. Jaké má výhody a nevýhody, a u jakého druhu zařízení ho asi najdete? Rozmyslete si, jak by pro takové zařízení mohl vypadat komunikační protokol burst zápisu, a nakreslete, jak by nějaký vhodný burst zápis tímto komunikačním protokolem u takového zařízení mohl typicky vypadat.

Otázka č. 10

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 naformátovaného souborovým systémem FAT16. Jako 2. argument na příkazové řádce (string z volání `ParamStr(2)`) program dostane číslo sektoru relativní k začátku obrazu, a zapsané v šestnáctkové soustavě. Víme, že takový sektor obsahuje data (pole záznamů) nějakého adresářového souboru, kde pro každý soubor v adresáři je zde 1 záznam v následujícím formátu:

Offset	|B|	Popis
`0x00`	8	ASCII řetězec se jménem souboru (zprava dorovnaný mezerami = ASCII kód `$20`). Pokud je `0`. byte roven `0x00` nebo `0xE5`, tak je tento adresářový záznam nepoužitý a byty `1`-`31` nenesou platné informace.
`0x08`	3	ASCII řetězec přípony souboru (zprava dorovnaný mezerami). Soubory bez přípony mají tuto položku vyplněnou 3 mezerami.
`0x0B`	1	Sada příznaků: bit 4: `0` = soubor, `1` = adresář
`0x0C`	10	Rezervováno
`0x16`	2	Čas poslední modifikace souboru
`0x18`	2	Datum poslední modifikace souboru
`0x1A`	2	Číslo prvního datového sektoru souboru
`0x1C`	4	Délka souboru v bytech

Program má vypsat jména všech souborů (a nikoliv podadresářů), které popisují záznamy z daného sektoru.