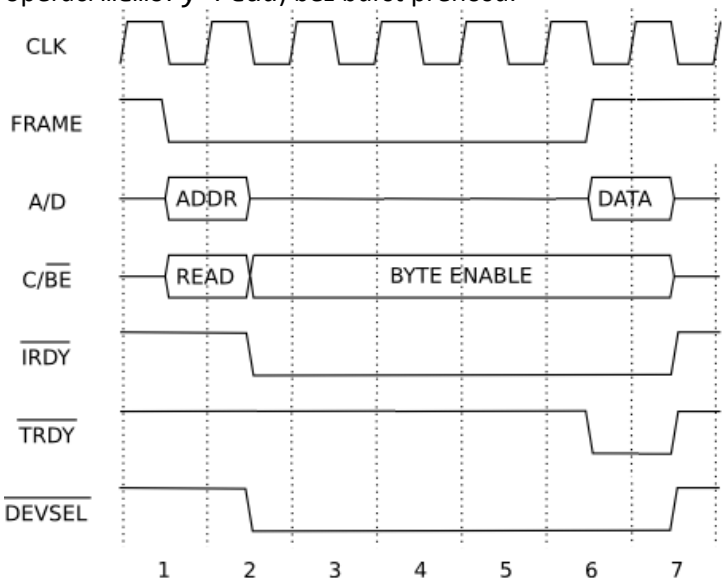
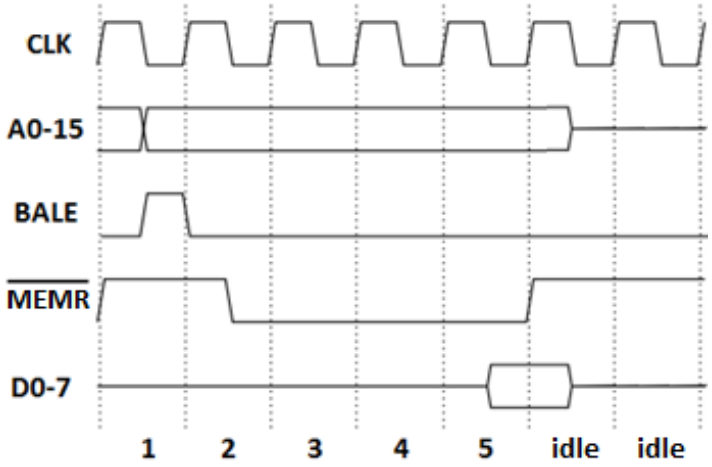
Odpovědi pište na zvláštní odpovědní list s vaším jménem a fotografií. Pokud budete odevzdávat více než jeden list s řešením, tak se na 2. a další listy nezapomeňte podepsat. Do zápatí všech listů vždy napište i/N (kde i je číslo listu, N je celkový počet odevzdaných listů).

Společná část pro otázky označené X

Předpokládejte, že máme počítač, kde je systémovou sběrnicí standardní 32-bitová varianta sběrnice PCI s následujícím standardním cyklem čtení z paměťového adresového prostoru (pokud signální vodiče C/$\underline{BE}_0$ až C/$\underline{BE}_3$ nabývají hodnoty `0110`, tak iniciátor transakce požaduje operaci `memory read`) bez burst přenosu:



Dále je na tuto systémovou PCI sběrnici pomocí PCI/ISA bridge připojena varianta ISA sběrnice (8-bitová datová cesta, podpora pro 16-bitový paměťový adresový prostor). Tato sběrnice má následující podobu zkráceného čtecího cyklu z paměťového adresového prostoru:



Pro jednoduchost předpokládejte, že obě uvedené sběrnice mají v našem počítači stejnou taktovací frekvenci. Dále předpokládejte, že zařízení na ISA sběrnici podporují libovolné protažení libovolného taktu přidržením hodinového signálu v úrovni `0` nebo `1`.

Otázka č. 1 za 0,5 bodu (X)

Popište a vysvětlete význam signálu BALE sběrnice ISA a proveďte srovnání se sběrnicí PCI, kde se tento signál nenachází. Dále vysvětlete význam signálů $\underline{TRDY}$ a $\underline{DEVSEL}$ u sběrnice PCI.

Otázka č. 2 za 1,5 bodu (X)

Předpokládejte, že pro uvedený počítač chceme navrhnout uvedený **PCI/ISA bridge** formou naprogramování 8-bitového MCU s harvardskou architekturou – bridge bude zjednodušený, a bude jen na ISA sběrnici předávat veškeré operace čtení z paměťového adresového prostoru z adres 0 až `FFFFh` iniciované na PCI sběrnici. Ostatní PCI transakce bude bridge ignorovat. Bridge bude fungovat jako master ISA sběrnice a jeho úkolem bude generovat adekvátní řídící signály včetně signálu hodinového (o stejném taktu jako u sběrnice PCI). Pro implementaci použijeme následující ***MCU*** se zabudovaným řadičem nezávislých programově ovladatelných digitálních vstupně/výstupních pinů MCU:

- Procesor má 8 šestnáctic vstupních pinů číslovaných `I0` až `I127`. Část HCI tohoto řadiče pro práci s těmito vstupními linkami tvoří osm 16-bitových registrů pouze pro čtení na adresách `100h` (pro čtení stavu pinů `I0` až `I15`) až `107h` (pro čtení stavu pinů `I111` až `I127`). Bit `0` každého takového registru odráží stav pinu s nejnižším číslem z určité šestnáctice, až bit 15 stav pinu s nejvyšším číslem (pokud je pin vně MCU zapojen na kladné napětí je v daném bitu hodnota 1, pro zem je v bitu hodnota `0`, pro nepřipojený pin je v bitu náhodná hodnota).
- Dále má procesor 8 osmic výstupních pinů číslovaných `O0` až `O63`. Druhou část HCI tohoto řadiče tvoří HCI pro práci s těmito výstupními linkami tvořící osm 16-bitových registrů (každý ovládající **jednu osmici** pinů) pouze pro zápis na adresách `200h` (pro nastavení stavu pinů `O0` až `O7`) až `207h` (pro nastavení stavu pinů `O55` až `63`). Pro každý z výstupních pinů jsou v registrech `O`*n* vyhrazeny 2 bity – horních 8 bitů registru rozhoduje o připojení výstupních pinů na napětí (`0` = floating stav daného pinu, 1 = pin připojen na `0`V nebo 5V), dolních 8 bitů rozhoduje o konkrétní hodnotě „vysílaného“ napětí (`0` = `0`V, 1 = 5V).

Úloha: Předpokládejte, že uvedený MCU připojíme následujícím způsobem ke sběrnicím:

- Ke sběrnici **PCI**: ▪ vstupní piny `I0`-`I31` na vodiče A/D_0-A/D_{31}, piny `I32`-`I35` na vodiče C/$\underline{BE}_0$-C/$\underline{BE}_3$, pin `I36` na vodič FRAME, pin `I37` na vodič $\underline{IRDY}$, pin `I38` na vodič CLK
  ▪ výstupní pin `O19` na vodič $\underline{TRDY}$, pin `O20` na vodič $\underline{DEVSEL}$, piny `O24`-`O31` na vodiče A/D_0-A/D_7
- Ke sběrnici **ISA**: ▪ vstupní piny `I40`-`I47` na vodiče D0-D7
  ▪ výstupní piny `O0`-`O15` na vodiče A0-A15, pin `O16` na vodič CLK, pin `O17` na vodič BALE, pin `O18` na vodič $\underline{MEMR}$.

Pro toto zapojení napište v Pascalu program firmware pro takový MCU tak, aby se celou dobu svého běhu choval jako výše popsaný bridge. Předpokládejte, že taktovací frekvence MCU je pro vaše potřeby dostatečně vysoká (řádově větší než frekvence použitých sběrnic PCI a ISA). Pro čekání na změnu stavu nějakého signálního vodiče sběrnice PCI používejte aktivní čekání (pollování). ***Doporučení:*** Pro větší přehlednost si v programu zaveďte pojmenované konstanty pro bity reprezentující hlavní kontrolní signály, případně pomocné funkce pro opakující se bloky kódu.

Otázka č. 3

Předpokládejte, že máme na USB flash disku nainstalovaný nějaký běžný OS. Tuto „flešku“ zapojíme do USB konektoru vypnutého notebooku IBM PC kompatibilního s 64-bit CPU Intel i5. Po zapnutí počítače dojde k nabootování do našeho OS. Popište a vysvětlete, co se děje, a jaký kód CPU zpracovává, v průběhu celého procesu bootování, tj. od zapnutí napájení až do začátku běhu kódu shellu OS.

Otázka č. 4

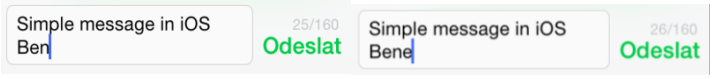
Předpokládejte operační systém s podporou pro vícevláknové zpracování a s preemptivním přepínám vláken. Jaké další (pokud nějaké) vlastnosti musí takový OS splňovat, aby v něm mohlo dojít k problému zvanému *priority inversion*? Na příkladu pak uveďte, kdy k takovému problému dochází, a navrhněte jeho řešení.

Otázka č. 5

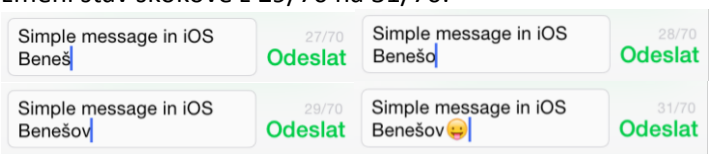
Předpokládejte, že implementujete jádro operačního systému určeného pro jednoprocesorové systémy a implementující preemptivní přepínání vláken s jednoduchým round robin plánovačem. Naimplementujte v Pascalu API funkci `WaitForLock` plánovače takového OS – tato funkce se využívá v implementaci zámků, a volá se, když se nějaké vlákno pokusí zamknout zámek vlastněný jiným vláknem. Můžete předpokládat, že po celý běh libovolné API funkce OS jsou zakázána všechna maskovatelná přerušení. Stručně popište a vysvětlete všechny datové struktury, globální proměnné, a procedury a funkce, které budete ve vaší implementaci používat. Do kódu své implementace zahrňte i proces, jakým se budou vlákna čekající „ve `WaitForLock`“ probouzet.

Otázka č. 6

Na telefonu iPhone s iOS 8.3 jsme ve standardní aplikaci pro posílání SMS zpráv napsali část takové zprávy – v pravé části je zobrazeno, že jsme použili 25 z celkovým 160 znaků podporovaných jednou SMS zprávou. Dále jsme dopsáním písmena „`e`“ zvýšili počet použitých znaků na 26 ze 160:



Po zapsání znaku „`š`“ se zobrazení změní na 27 ze 70 znaků (tedy se pro celou zprávu změnilo kódování všech znaků ve zprávě – jak ukazuje i zapsání dalších znaků „`o`“ [stav 28/70] a „`v`“ [stav 29/70]). Navíc se po zapsání znaku „smajlík“ změní stav skokově z 29/70 na 31/70:



Vysvětlete a zdůvodněte, jaké se asi používá kódování pro 160 znakové SMS zprávy, a jaké pro 70 znakové SMS zprávy. Dále vysvětlete, proč jeden znak „smajlík“ zabral ve skutečnosti dva znaky v 70 znakové zprávě? Vysvětlete, v jakém rozsahu se asi nachází kód znaku „smajlík“: `0-127`, `128-255`, `256-1023`, `1024-$FFFF`, nebo `$10000-$FFFFFFFF`?

Otázka č. 7

Vyrábí se dnes ještě běžně procesory s tzv. akumulátorovou architekturou? Pokud ano, tak vysvětlete proč, a u jakého typu procesoru byste ji očekávali. Pokud ne, tak vysvětlete proč. Jaké jsou hlavní výhody a nevýhody akumulátorové architektury?

Otázka č. 8

Původní počítač IBM PC obsahoval 16-bitový procesor Intel 8088 s podporou pro 20-bitový adresový prostor a bez podpory pro stránkování. Pro podporu více než `1 MB` operační paměti bylo možné do počítače zapojit rozšiřující kartu tzv. systému EMS (Expanded Memory Specification). EMS karta obsahovala dalších až `16 MB` paměti RAM, a separátní řadič paměti, který pomocí mechanizmu bank switchingu na adresách `0xE0000–0xEFFFF` zpřístupňoval vždy jednu z `256` bank rozšířené paměti EMS. Běžně používaný OS na tomto počítači byl MS-DOS a uměl tohoto mechanizmu využít.

Díky režimu zpětné kompatibility lze operační systém MS-DOS spustit i na modernějším počítači s nainstalovanými `128 MB` operační paměti a s plně 32-bitovým procesorem Intel 80386, který podporuje stránkování se stránkami o velikosti `4096 B` (MS-DOS koncept stránkování nezná a při jeho startu a běhu je implicitně stránkování vypnuté). Jelikož ale kód jádra MS-DOS obsahuje pouze instrukce Intel 8088 kompatibilní pracující jen s `20`-bitovými adresami, máme i zde k dispozici jen spodní 1 MB operační paměti. Rozhodli jsme se proto pro MS-DOS naprogramovat ovladač sloužící jako emulátor EMS rozšiřujících karet, který operačnímu systému zpřístupní nainstalovanou operační paměť z rozsahu `1 MB – 17 MB` po stejných `64 kB` „bankách“ jako u původní paměti EMS. Celý systém musí být pro MS-DOS transparetní a chovat se jako původní IBM PC se zasunutou EMS kartou. Vysvětlete, jak byste uvedený ovladač naprogramovali (jak by se choval) s využitím zapnutí stránkování procesoru 80386. Detailně popište, jakým obsahem v ovladači vyplníte stránkovací tabulku, a zda budete její obsah někdy měnit!

Otázka č. 9

Předpokládejte běžný počítač se zabudovaným řadičem pevných disků. Takový řadič může na CPU způsobit vznik hardwarového přerušení. Uveďte příklad situace, kdy by běžný řadič pevných disků inicioval vznik takového přerušení. Bude se jednat o přerušení synchronní nebo asynchronní? Co je obsahem tabulky vektorů přerušení? Kdo typicky obsah tabulky vyplňuje a kdo její obsah čte?

Otázka č. 10

Předpokládejte nějaký typický procesor s obecnou registrovou architekturou. Jaké typické příznaky (*flags*) takový procesor bude mít (uveďte alespoň 3 nejběžnější), a kde jsou uloženy? Pro každý uvedený příznak vysvětlete, co znamenají hodnoty, kterých může nabývat, a dále vysvětlete, jaké instrukce budou takový příznak měnit, resp. číst. Dále pro každý uvedený příznak uveďte jednu typickou situaci, kdy se v nějakém programu existence takového příznaku využívá.