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 α**

Předpokládejte, že máme sběrnici, která je variantou původní 8-bitové sběrnice ISA, tedy paralelní sběrnice nepodporující burst přenosy, ani DMA bus mastering. Navíc má ale naše varianta následující odlišnosti: podpora pouze pro 8-bitový paměťový adresový prostor, a pro 8-bitový I/O adresový prostor, základní taktovací frekvence je 4,77 Mhz.

---

**Otázka č. 1 (α)**

Nakreslete časový diagram průběhu hodnot na všech důležitých vodičích v průběhu jedné celé transakce zápisu hodnoty 13 na adresu 255 v I/O adresovém prostoru. Nakreslete průběh pro každý takový vodič zvlášť, vodiče neseskupujte. K vašemu obrázku napište krátkou legendu, tj. pro každý použitý vodič vysvětlete jeho význam.

---

**Otázka č. 2 (α)**

Předpokládejte, že pro připojení k uvedené ISA sběrnici chceme navrhnout řadič pro ovládání otáček větráčku chladícího chipset základní desky:

***Řadič větráčku*** má mít jeden 8-bitový port-mapped registr na adrese FFh. Zápisem do toho registru určujeme počet otáček větráčku za minutu – hodnota 0 znamená úplné zastavení, hodnota 127 znamená maximální otáčky podporované větráčkem (hodnoty mimo tento rozsah zapsané do registru na adrese FFh se interpretují jako hodnota 127. K našemu řadiči bude připojen větráček, který se ovládá jedním signálním vodičem s PWM (Pulse Width Modulation) modulací (nulové PWM plnění znamená zastavený větráček, maximální PWM plnění znamená maximální otáčky). Řadič chceme vyrobit formou naprogramování 16-bitového jednočipového počítače s harvardskou architekturou a s podporou pro 8 vektorů přerušení:

***Mikrokontroler (MCU)*** má zabudovaný výstupní PWM (Pulse Width Modulation) modulátor (výstup z pinu procesoru č. 32). Tento PWM modulátor (vlastně varianta D/A převodníku) používá jeden paměťově mapovaný W/O 8-bitový registr na adrese 206h. Hodnoty zapsané do tohoto registru jsou lineálním mapováním převedeny na PWM signál s daným plněním – hodnota 0 zapsaná do registru znamená generování PWM signálu s nulovým plněním na pinu 32, hodnota 255 znamená generování PWM signálu s maximálním plněním.

Dále má MCU v sobě zabudovaný řadič 4 krát 8 (tj. celkem 32) nezávislých vstupních GPIO linek (programově čitelných digitálních vstupních pinů procesoru číslovaných 0 až 31). HCI tohoto řadiče GPIO tvoří čtyři 8-bitové registry a 4 IRQ:

- R/O registr INR1 (na adrese 141h) slouží ke čtení stavu 1. osmice pinů, tj. ke čtení 8 jednobitových hodnot z pinů 0 (bit 0 tohoto registru) až 7 (bit 7).
  Čtením z tohoto registru získáme aktuální stav na všech pinech 0 až 7 (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).
- Podobně fungují i registry INR2 (na adrese 142h) pro piny 8 až 15 (opět bit 0 = pin 8, až bit 7 = pin 15), INR3 (na 143h) pro piny 16 až 23, INR4 (na 144h) pro piny 24 až 31.
- Při změně stavu libovolného bitu (hodnoty) registru INR1 generuje GPIO řadič v mikrokontroleru interní žádost o přerušení číslo 1, při změně stavu INR2 generuje přerušení 2, apod. pro INR3 IRQ3, až pro INR4 IRQ4.

***Úloha:***

Předpokládejte, že uvedený MCU připojíme následujícím způsobem ke sběrnici ISA: piny 0-7 na vodiče A0-A7, piny 8-15 na vodiče D0-D7, pin 16 na vodič CLK, pin 21 na vodič IOW, pin 23 na vodič BALE. Dále bude pin 32 připojen na PWM vstup větráčku.

Pro toto zapojení napište v Pascalu program firmware pro takový MCU tak, aby se celou dobu svého běhu choval jako výše popsaný řadič větráčku, kdy po spuštění MCU se má větráček otáčet maximálními otáčkami. Na MCU ale bude běžet ještě i jiná funkcionalita, kterou budeme programovat až později, proto váš „program“ (algoritmus) nepište do „hlavního programu“ Pascalu, ale zapište ho do následujících procedur (libovolnou z nich můžete nechat prázdnou, můžete si k nim deklarovat libovolné množství globálních proměnných):

```
procedure Inicializace;
procedure ObsluhaIRQ(cisloIrq : Interger);
```

Kde vaše procedura `Inicializace` bude vyvolána jednou při bootu mikrokontroleru, procedura `ObsluhaIRQ` je volána při každém vzniku přerušení 1 až 4, jehož číslo získáte v parametru `cisloIrq`. Předpokládejte, že taktovací frekvence MCU je pro vaše potřeby dostatečně vysoká (řádově větší než frekvence sběrnice ISA). Pro čekání na změnu stavu nějakého signálního vodiče sběrnice ISA **nepoužívejte** aktivní čekání (pollování). ***Doporučení:*** Pro větší přehlednost si v programu zaveďte pojmenované konstanty pro bity reprezentující hlavní kontrolní signály.

---

**Otázka č. 3**

Předpokládejte, že chcete pro mobilní telefony programovat větší množství různých aplikací, které všechny budou využívat dat ze zabudovaných akcelerometrů a gyroskopů. Každou z naprogramovaných aplikací budete chtít v binární spustitelné podobě distribuovat pro operační systémy iOS (běžící na iPhone) a Windows Phone 8. Oba tyto operační systémy mají ale rozdílné API (navzájem nekompatibilní) pro komunikaci s akcelerometry a gyroskopy, a zároveň programovací jazyk, který budete používat pro programování aplikací, nemá žádnou podporu pro přímou práci s takovými zařízeními. Aplikace chcete programovat tak, aby byly přenositelné na úrovni zdrojového kódu mezi oběma uvedenými systémy. Jakým způsobem to nejlépe zařídíte? Jakým způsobem pak bude probíhat překlad veškerého potřebného kódu každé takové aplikace, až do stavu, kdy máme finální spustitelné soubory?

---

### Otázka č. 4

***Úloha:*** Čistě v assembleru níže uvedeného 32-bitového procesoru naimplementujte kompletní níže uvedenou funkci `f` při použití varianty Cčkové volací konvence (parametry se předávají na zásobníku zprava doleva, parametry odstraňuje volající, návratová hodnota se vrací v registru EAX) a za předpokladu, že `Longword` je celočíselný bezznaménkový 32-bitový typ:

```
function f(a : Longword) : Longword; begin
   if a = 1 then begin
      f := 1;
   end else if a = 2 then begin
      f := 1;
   end else begin
      f := f(a - 1) + f(a - 2);
   end;
end;
```

***Instrukční sada:*** Kód implementujte pro počítač s variantou 32-bitového procesoru Intel 80386 běžícího v 32-bitovém režimu s vypnutou podporou pro stránkování (virtuální i fyzický adresový prostor procesoru má šířku 32-bitů, virtuální adresa se přímo rovná adrese fyzické). Procesor má obecnou registrovou architekturu, a mimo jiné 7 obecných 32-bitových registrů `EAX`, `EBX`, `ECX`, `EDX`, `EBP`, `EDI`, `ESI`, dále má 32-bitový registr `ESP` (stack pointer), a 32-bitový registr `EIP` (program counter), a příznakový registr se všemi běžnými příznaky. Můžete využít instrukce `MOV`, `ADD`, `SUB` (odečítání), `MUL` (násobení), `CMP`, `JZ`, `JNZ`, `JMP`, `PUSH`, `POP`, `CALL`, `RET`, které mají všechny standardní chování. Instrukce ovlivňují příznaky standardním způsobem, aritmetické instrukce jsou dvouoperandové. Pro zápis vašeho kódu využijte běžné Intel syntaxe, tedy: cílový operand instrukce je vždy nejvíce vlevo; immediate hodnota je libovolné číslo bez prefixu; hodnota v hranatých závorkách znamená variantu instrukce s operandem typu adresa, tj. např. `[x]` znamená hodnotu operandu na adrese `x`; běžné instrukce procesoru 80386 mají i variantu, kde se adresa operandu spočítá jednoduchým výpočtem, a tedy např. zápis `[x+y]` znamená operand ležící na adrese, která vznikne součtem hodnot `x` a `y`, kde `x` i `y` mohou být libovolný registr, pouze `y` může být i hodnota immediate, u instrukcí s více operandy smí být pouze jeden adresa v `[]`. Cíl skoku si můžete v assembleru označit jako label podobně jako v Pascalu, tedy libovolné jméno s dvojtečkou.

***Př.:*** pokud je v registru `EDI` hodnota `$00AA0050`:

```
něco:                 { následující instrukce mov je cílem skoku }
  mov eax, [edi+4]    { načtení 4B hodnoty z adresy $00AA0054
                        a její uložení do registru EAX }
  jmp něco            { proveď skok na adresu instrukce za
                        lablem něco, tj. zde na instrukci mov }
```

---

### Otázka č. 5

Předpokládejte, že v Linuxu na příkazovou řádku napíšeme text `vim` a stiskneme klávesu `Enter` – následovat bude spuštění textového editoru Vim. Detailně popište a vysvětlete, jaký kód bude reagovat na stisk klávesy Enter, a jaké všechny kroky bude třeba provést, a jaké části kódu je budou v OS a jeho jádře provádět, od stisku klávesy Enter do začátku běhu hlavního vlákna editoru Vim.

---

### Společná část pro otázky označené β

Předpokládejte běžný moderní desktopový počítač vyrobený po roce 2013 s procesorem Intel i7 nebo podobným, do jehož zadního USB konektoru jsme připojili standardní USB klávesnici.

---

### Otázka č. 6 (β)

Detailně popište, jaká je vnitřní architektura takového počítače, po jakých všech sběrnicích proudí data mezi ovladačem klávesnice a uvedenou USB klávesnicí, a co případně zařizuje propojení takových sběrnic. K výkladu nakreslete ilustrativní obrázek, na kterém vyznačte a popište veškerá zařízení a části sběrnic po cestě.

---

### Otázka č. 7 (β)

Předpokládejte, že na daném počítači máme nainstalovaný nějaký moderní operační systém, na němž spustíme program napsaný v Pascalu, který během svého běhu zavolá funkci `ReadKey`. Po stisku klávesy Q na uvedené USB klávesnici se z funkce `ReadKey` do programu vrátí kód stisknuté klávesy. Na ilustrativním obrázku popište, přes které všechny části kódu v aplikaci, OS, a jeho jádře se předává informace o stisku klávesy od chvíle, kdy je vyslána samotnou klávesnicí. Ke každé části napište její stručný popis. Zaměřte se hlavně na strukturu driver stacku a jeho částí.

---

### Otázka č. 8

Předpokládejte, že máte binární soubor, jehož data jsou v souboru uložena jako little endian, a že od `0`. bytu je v souboru uloženo 64-bitové reálné číslo s pohyblivou desetinnou čárkou ve formátu IEEE 754 double, tj. mantisa je normalizována se skrytou `1` a zabírá spodních `52` bitů, pak následuje `11`-bitový exponent uložený ve formátu s posunem (bias) `+1023`, a `1` znaménkový bit. Hodnota tohoto čísla je `4093,25` zapsáno v desítkové soustavě. Zapište v šestnáctkové soustavě hodnotu prvních `8` bytů takového souboru.

---

### Otázka č. 9

Předpokládejme, že v jádře operačního systému poskytujícího podporu pro vícevláknové zpracování chceme v Pascalu naimplementovat proceduru `Join(TID : Longint)`, která způsobí, že vlákno, které ji zavolá, nebude ve zpracování dalších instrukcí pokračovat dříve, než do konce své hlavní procedury doběhne jiné vlákno identifikované `TID`. Vyberte pro zmíněný OS se souběžným během mnoha aplikací tu nejvhodnější implementaci daného čekání, a v Pascalu zapište kód procedury `Join` a všech ostatních částí OS, které budou pro její fungování potřeba (soustřeďte se jen na části související se samotným procesem od začátku do konce čekání volajícího vlákna).

---

### Otázka č. 10

Vysvětlete, co to je tzv. UTF-8, jak funguje a proč se používá. Popište též jeho výhody a nevýhody vzhledem k „mechanismům“ s podobnou funkcí.