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 následující program napsaný v Pascalu a přeložený Free Pascal překladačem do 32-bitového kódu procesoru Intel i7 (ISA x86) a pro operační systém Windows 7 32-bit:

```
procedure Foo; cdecl;
begin
   Writeln('This is Foo!');
end;

procedure Bar(a : longword); cdecl;
begin
   {*}
   Writeln('This is Bar!');
end;

begin
   Writeln('Main program: before Bar ...');
   Bar(1);
   Writeln('Main program: after Bar ...');
end.
```

Dále předpokládejte, že v místě označeném ***{*}*** dopíšeme kód, který za běhu procedury `Bar` přepíše aktuální návratovou adresu z `Bar` do hlavního programu adresou procedury `Foo`. Po skončení volání procedury `Bar` tedy procesor pokračuje ve zpracování instrukcí procedury `Foo`. Klíčové slovo **`cdecl`** přikazuje překladači použít běžnou volací konvenci jazyka C, tedy, že parametry procedur a fcí se předávají na zásobníku zprava doleva, návratová hodnota se předává v registru `EAX`, volaný musí volajícímu zachovat minimálně obsah registru `EBP`.

Otázka č. 1 (X)

Dopište do místa označeného ***{*}*** kód v Pascalu, který se bude chovat dle výše uvedeného popisu (tj. změní návratovou adresu z procedury `Bar`). Kód za místem ***{*}*** se při volání procedury `Bar` musí stále provést a ve vašem řešení do něj nesmíte zasahovat. Typ `longword` je v Pascalu celé bezznaménkové 32-bitové číslo.

Otázka č. 2 (X)

Za předpokladu kontextu uvedeného ve společné části detailně vysvětlete, co a proč se bude dít (jaký kód a proč bude CPU vykonávat) po spuštění *upraveného* programu po tom, co procedura `Foo` doběhne do konce.

Otázka č. 3 (X)

Bylo by možné uvedený program beze změny přeložit i pro operační systém Linux pro 32-bitové procesory ARM? Pokud ano, tak vysvětlete, jak by překlad probíhal a zda, případně jak, by se lišil výsledný spustitelný soubor od varianty pro Windows. Pokud ne, tak detailně vysvětlete proč, a jaké změny bychom v programu museli provést.

Společná část pro otázky označené Y

Kódování UTF-8 umožňuje zakódovat libovolný Unicode znak z rozsahu U+0000 až U+10FFFF. Pokud je znak v UTF-8 kódován delší než 1 bytovou sekvencí, tak počet bytů jednoho znaku zjistíte z 1. bytu takové sekvence: pro každý N-bytový znak (kde N > 1) platí, že nejvyšších N bitů 1. bytu je rovno `1`, pak následuje jeden bit rovný `0`. Druhý až N-tý byte sekvence vypadají tak, že bit `7` je roven `1`, bit `6` je roven `0`. Pro N-bytové sekvence pro N > 1 obsahuje zbývajících 5/4/3 bitů 1. bytu a vždy zbývajících 6 bitů dalších bytů rozdělenou informaci o skutečném kódu znaku – viz bity x v následujícím schématu:

```
  110xxxxx 10xxxxxx
  1110xxxx 10xxxxxx 10xxxxxx
  11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
```

Dvoubytovou sekvencí je kódována právě souvislá řada Unicode znaků s kódy přímo následujícími po znacích kódovaných jedním bytem. Ve dvoubytové sekvenci obsahuje 11 bitů označených x rovnou původní kód znaku od vyšších řádů dorovnaný nulami. Stejně tak 16 bitů x tříbytové sekvence kóduje opět další následující souvislou řadu Unicode znaků, a 21 bitů x čtyřbytové sekvence kóduje poslední souvislou řadu znaků až do posledního kódu `0x10FFFF`. V dokumentu RFC 3629 jakožto specifikaci UTF-8 je dále napsáno: „Fill in the bits marked x from the bits of the character number, expressed in binary. Start by putting the lowest-order bit of the character number in the lowest-order position of the last octet of the sequence, then put the next higher-order bit of the character number in the next higher-order position of that octet, etc. When the x bits of the last octet are filled in, move on to the next to last octet, then to the preceding one, etc. until all x bits are filled in."

Otázka č. 4 (Y)

Napište v šestnáctkové soustavě všechny čtyři rozsahy Unicode znaků, které jsou kódovány 1, 2, 3, a 4-bytovými sekvencemi v UTF-8. Dále v šestnáctkové soustavě zapište hodnoty všech bytů kódujících v UTF-8 nulou ukončený řetězec „Čau" bez uvozovek, když víte, že znak `A` má v ASCII kód `$41`, znak a má v ASCII kód `$61`, a combining znak háček má v Unicode kód `$30C`.

Otázka č. 5 (Y)

Naprogramujte v Pascalu s vhodným využitím bitových operací co nejefektivnějším způsobem následující proceduru, která má převést vstupní null-terminated řetězec `in` v kódování UTF-32 do kódování UTF-8 (předpokládejte, že proměnná `out` ukazuje na dostatečně velkou předalokovanou paměť, kam se vám celý výsledný UTF-8 null-terminated řetězec vejde):

```
type
  PUtf8 = ^byte;
  PUtf32 = ^longword;
procedure From32To8(in : PUtf32; out : PUtf8);
```

Otázka č. 6

Předpokládejte, že pro společnost Cypress navrhujeme I²C bus interface 512-Kbit (64K x 8) NVRAM paměti s typovým označením FM24V05. Paměťový modul je navržený tak, že kromě standardních 2 pinů pro připojení k I²C sběrnici má také 3 piny `A0` až `A2` pro určení unikátní adresy paměti na jednom segmentu sběrnice I²C. Navrhněte a vysvětlete, jak by mohl vypadat komunikační protokol pro I²C pro čtení dat z takovéhoto paměťového modulu. S použitím tohoto protokolu napište příklad bitové posloupnosti, která by se přenesla po I²C sběrnici při čtení z adresy `0xA90F`, pokud je na této adrese v paměti uložena hodnota `0x42` (označte, který bit se bude přenášet jako první; do posloupnosti zaneste i speciální „stavy"/ „symboly" indikované na sběrnici; pro každý bit označte, kdo ho bude vysílat; také označte případné související skupiny bitů). Pokud vám k řešení v zadání chybějí nějaké informace, tak si je v kontextu vhodně zvolte a v řešení je uveďte.

Otázka č. 7

Detailně vysvětlete, k čemu na I²C sběrnici slouží tzv. *repeated start*, a zda by bylo možné se bez něj obejít. Jakým způsobem se *repeated start* rozpozná od přenosu běžného bitu?

Společná část pro otázky označené Z

Předpokládejte, že máme 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/logická adresa se přímo rovná adrese fyzické), procesor kromě paměťového prostoru podporuje ještě 16-bit I/O adresový prostor. Procesor má obecnou registrovou architekturu, a mimo jiné 7 obecných 32-bitových registrů `EAX`, `EBX`, `ECX`, `EDX`, `ESI`, `EDI`, `EBP`, (spodních 16 bitů registrů `EAX` až `EDX` je přístupných pomocí 16-bit „pseudoregistrů" `AX`, `BX`, `CX`, `DX`, a jejich nejnižších 8 bitů je dále přístupných pomocí 8-bit „pseudoregistrů" `AL`, `BL`, `CL`, `DL`), dále má 32-bitový registr `ESP` (stack pointer), 32-bitový registr `EIP` (program counter), a příznakový registr s běžnými příznaky. ISA obsahuje instrukce `MOV` (má 3 varianty: load, store, přesun mezi dvěma registry), `PUSH`, `POP`, `ADD` (sčítání bez přenosu), `SUB` (odečítání bez přenosu), `JNZ` (Jump if Not Zero), `CMP` (compare), `JMP`, `CALL`, a `RET` s běžnou sémantikou, dále má instrukce `IN AL,DX` (načtení 1 byte do registru `AL` z adresy v I/O adresovém prostoru dané obsahem registru `DX`), a `OUT DX,AL` (zápis 1 byte z `AL` na I/O adresu v `DX`). Instrukce `MOV`, `ADD`, `SUB`, `CMP` mají dva argumenty (maximálně jeden smí být typu adresa – viz níže), `PUSH`, `POP`, `JNZ`, `JMP`, `CALL` mají jeden argument, instrukce `RET` je bez argumentů. Argumentem uvedených instrukcí (s výjimkou instrukcí `IN` a `OUT`, které mají pevně dané operandy, viz výše) může být immediate (kde to dává smysl), libovolný obecný registr nebo registr `ESP`, nebo adresa (formát viz níže).

Intel syntaxe assembleru pro uvedený procesor je následující: cílový operand instrukce je vždy nejvíce vlevo; immediate operand je číslo bez prefixů v desítkové nebo šestnáctkové soustavě (přípona `h`), hodnota v hranatých závorkách je operand typu adresa, tj. např. `[imm/reg]` znamená hodnotu operandu na adrese `imm/reg`, což je nejjednodušší varianta nejobecnějšího procesorem podporovaného operandu typu adresa: `[reg1 + (reg2 * imm2) + imm1]`, který znamená hodnotu na adrese spočítané jako výsledek výrazu v hranatých závorkách. Platí: `imm` jsou immediate, `imm2` může být pouze `1`, `2`, `4`, nebo `8`, `reg` může být obecný registr nebo registr `ESP`. Všechny části výrazu jsou nepovinné.

Otázka č. 8 (Z)

Napište v Pascalu bez použití inline assembleru kód fce (i s deklarací), která by mohla být běžným překladačem přeložena do kódu disassemblovaného od adresy `0x01363C30` uvedeného počítače – víme, že použitý překladač Pascalu používá variantu Cčkové volací konvence, kdy jsou argumenty funkce ukládány na zásobník zprava doleva, návratová hodnota je v registru `EAX`, volaný musí volajícímu zachovat minimálně obsah registrů `EBP` a `ESI`):

```
01363C30 push ebp
01363C31 mov  ebp,esp
01363C3A push esi
01363C4E cmp  dword ptr [ebp+08h],0
01363C52 jnz  01363C5Ah
01363C54 mov  eax,0
01363C56 jmp  01363C8Ch
01363C5A cmp  dword ptr [ebp+08h],1
01363C5E jnz  01363C69h
01363C60 mov  eax,1
01363C67 jmp  01363C8Ch
01363C69 mov  eax,dword ptr [ebp+08h]
01363C6C sub  eax,1
01363C6F push eax
01363C70 call 01363C30h
01363C75 add  esp,4
01363C78 mov  esi,eax
01363C7A mov  ecx,dword ptr [ebp+08h]
01363C7D sub  ecx,2
01363C80 push ecx
01363C81 call 01363C30h
01363C86 add  esp,4
01363C89 add  eax,esi
01363C8C pop  esi
01363C9D pop  ebp
01363C9E ret
```

Otázka č. 9 (Z)

Předpokládejte, že programujeme část jádra OS určeného pro uvedený počítač, která má za úkol detekovat zařízení připojená na PCI sběrnici v počítači a podle ID zařízení nahrát do paměti vhodný ovladač zařízení. Naprogramujte v Pascalu následující funkci, která má přečíst 4 bytovou hodnotu z adresy `addr` v **konfiguračním** adresovém prostoru PCI sběrnice (pokud byste potřebovali znát adresy registrů nějakých zařízení, tak si je vhodně zvolte) – své řešení okomentujte a zdůvodněte:

```
function ReadPciCfg(addr : longword) : longword;
```

Otázka č. 10

Mějme DRAM čip s organizací 1 M x 16. Jaké signální vodiče paměť bude mít? Proč? Jaký je rozdíl mezi DRAM a SRAM?