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 I²C sběrnici s referenční frekvencí 100 kHz, se 7-bitovými adresami, a MSB-first přenosem dat. Pokud na sběrnici neprobíhá žádný přenos, tak jsou signály SDA i SCL ve vysoké úrovni. Začátek přenosu je signalizován přechodem SDA do nízké úrovně, který je po nějaké době následován přechodem SCL do nízké úrovně. Pak následuje přenos dat – data jsou platná při vysoké úrovni hodinového signálu. Po přenosu posledního bitu musí SCL i SDA přejít do nízké úrovně – pak následuje označení konce přenosu: nejprve přechod SCL do vysoké úrovně, pak po nějaké době následovaný přechodem SDA do vysoké úrovně. Nyní je sběrnice opět volná. Přičemž si uvědomte, že během samotného přenosu při vysoké úrovni SCL ke změně úrovně signálu SDA nikdy dojít nemůže (nesmí).

Otázka č. 1 za 0,5 bodu (X)

Pro uvedenou I²C sběrnici nakreslete časový diagram kompletního přenosu (včetně výše popsaných značek začátku a konce přenosu) jednoho bytu s hodnotou $2 ze zařízení na adrese $B. Označte, kdo a v jakých rozsazích časů „generuje“ signál na jednotlivých vodičích této I²C sběrnice.

Otázka č. 2 za 1,5 bodu (X)

Pro uvedenou I²C sběrnici chceme vyrobit zařízení fungující jako digitální teploměr. Teploměr má fungovat pouze jako slave na jedné, ale „programovatelné“, adrese (tj. adresa není v zařízení napevno zabudována, ale dá se zvolit při zapojení do konkrétního obvodu). Při čtení z této adresy má teploměr vrátit jedno číslo od 0 do 255, kde 0 reprezentuje teplotu -30°C, až hodnota 255 teplotu +60°C. Takovéto zařízení budeme vyrábět formou naprogramování 8-bitového MCU s harvardskou architekturou:

MCU má zabudovaný A/D převodník (vstup z pinu procesoru č. 20). Tento A/D převodník používá jeden paměťově mapovaný R/O 8-bitový registr na adrese 100h. Hodnoty přečtené z tohoto registru jsou lineálním mapováním z hodnoty napětí na pinu 20 do celých čísel v rozsahu 0 až 255, kde hodnota 0 odpovídá napětí 0V na pinu 20, hodnota 255 odpovídá napětí 5V.

Dále má MCU zabudovaný řadič 2 krát 8 (tj. celkem 16) nezávislých GPIO linek (programově ovladatelných digitálních vstupně/výstupních pinů procesoru číslovaných 0 až 15). Pro každou z těchto dvou osmic GPIO pinů má řadič vyhrazenu jednu 8-bitovou výstupní paměť SRAM. HCI tohoto řadiče GPIO tvoří tři 8-bitové registry:

- R/W registr STATE (na adrese 95h) slouží k přepínání mezi režimy vstup nebo výstup pro celé osmice GPIO linek (bit 7 určuje režim 1. osmice, bit 6 určuje režim 2. osmice: hodnota 0 určitého bitu [počáteční hodnota po restartu MCU] = režim vstup pro danou osmici, hodnota 1 = režim výstup). Pokud je daná osmice v režimu vstup, tak je všech 8 pinů dané osmice odpojených (floating) ze strany MCU. Pokud je daná osmice v režimu výstup, tak je všech 8 pinů osmice připojených na kladné napěti = 1 nebo zem = 0 (dle obsahu registru IO*n* pro danou osmici *n*, viz dále).
- R/W registr IO1 (na adrese 90h) slouží k ovládání 1. osmice pinů, tj. ke čtení/zápisu 8 jednobitových hodnot z/do pinů 0 (bit 0) až 7 (bit 7).
  Čtením z tohoto registru (jen pokud je osmice v režimu vstupu) 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).
  Zápisem do tohoto registru (jak v režimu výstupu tak i v režimu vstupu této 1. osmice) se mění obsah 1 bytové výstupní paměti řadiče pro tuto osmici. V režimu výstupu je obsah výstupní paměti této osmice „vysílan“ MCU na pinech 0 až 7.

Podobně funguje i registr IO2 (na adrese 91h) pro piny 8 až 15 (opět bit 0 = pin 8, až bit 7 = pin 15).

Úloha: Předpokládejte, že uvedený MCU zapojíme následujícím způsobem: piny 0 až 6 určují bit 0 až bit 6 adresy, na které má teploměr poslouchat na I²C sběrnici (a tedy je napevno připojíme na kladné nebo nulové napětí dle zvolené adresy – adresa se po zapnutí MCU nemění), pin 7 připojíme na vodič SCL, pin 8 na vodič SDA, a k pinu 20 připojíme analogové teplotní čidlo (spojitě „generující“ pro teplotu -30°C napětí 0V, až pro teplotu +60°C napětí 5V). Pro toto zapojení napište v Pascalu program firmware pro takový MCU tak, aby se celou dobu svého běhu choval jako výše popsaný digitální teploměr. Předpokládejte, že taktovací frekvence MCU je pro vaše potřeby dostatečně vysoká (řádově větší než frekvence sběrnice I²C). Pro čekání na změnu stavu nějakého signálního vodiče sběrnice I²C používejte aktivní čekání (pollování). ***Doporučení:*** Pro větší přehlednost si v programu zaveďte pojmenované konstanty pro bity reprezentující hlavní kontrolní signály.

Otázka č. 3

Procesor Intel 80386 má kromě instrukce `ret` i speciální instrukci návratu z přerušení `iret`. Detailně vysvětlete, co instrukce `iret` dělá, a proč je potřeba. Do vysvětlení zahrňte, zda by bylo možné ji v programu nahradit posloupností jednodušších instrukcí. Proč to je/není možné?

Otázka č. 4

Popište všechny typické stavy, ve kterých může být vlákno v běžném systému s vícevláknovým zpracováním. Popište také všechny běžné přechody mezi jednotlivými stavy vlákna, a vysvětlete, v jaké situaci k danému přechodu dochází (co a kdy může být jeho příčinou).

Otázka č. 5

Mějme aplikaci napsanou v Pascalu, jejíž zdrojový kód je rozdělen do několika samostatných souborů (`a.pas` až `d.pas`, kde `d.pas` obsahuje hlavní tělo programu). Detailně vysvětlete, jak a v jakých krocích vznikne výsledný spustitelný soubor aplikace při použití typického překladače Pascalu. Na jakou instrukci bude ukazovat jeho *entrypoint*?

Otázka č. 6

Pouze s využitím celočíselné aritmetiky (Pascal typů) naprogramujete v Pascalu následující funkci `Mul`, která bude součástí RTL Pascalu jako podpora pro CPU bez instrukcí pro práci s čísly s pohyblivou desetinnou čárkou:

```
function Mul(f : Longword;
             n : Integer) : Longword;
```

Parametr f i návratová hodnota fce reprezentují 32-bit reálná čísla ve formátu IEEE 754 single (mantisa je normalizována se skrytou 1 a zabírá spodních 23 bitů, pak následuje 8-bitový exponent uložený ve formátu s posunem [bias] +127 a 1 znaménkový bit). Funkce má vrátit hodnotu `f` vynásobenou n-tou mocninou 2 (tedy $f*2^n$) pro n kladná i záporná. Okrajové situace (příliš malé n, příliš velké n, apod.) nemusíte řešit.

Otázka č. 7

Předpokládejte, že je naším úkolem v Pascalu naprogramovat JVM (Java Virtual Machine). Napište v Pascalu zjednodušený základ takové VM s harvardskou a se zásobníkovou registrovou architekturou (všechny registry obsahují 32-bit signed celá čísla) jako interpret Java bytecode pro následující 4 instrukce (vše jednobytový opcode následovaný případnými argumenty)

- `iconst_1` (opcode `$04`) – load constant 1 (bez arg.)
- `iconst_4` (opcode `$07`) – load constant 4 (bez arg.)
- `iadd` (opcode `$60`) – součet (bez arg.)
- `ifne` (opcode `$9A`) – podmíněný skok: instrukce odebere ze zásobníku jednu hodnotu, a pokud není rovna `0`, tak se provede skok na adresu, která je argumentem instrukce. Za opcodem vždy následují 2 byty, kde je v little endian pořadí zapsána cílová adresa skoku (od `0` do `65535`).

K dispozici máte implementaci zásobníku formou jednosměrně vázaného seznamu `Longint` hodnot, viz kód níže. Prázdný zásobník je reprezentován hodnotou `nil`. Procedura `Push` vkládá novou hodnotu na vršek zásobníku daný parametrem `top`, a do `top` nastaví nový vršek zásobníku. Funkce `Pop` vrací hodnotu z vršku zásobníku a vršek v `top` aktualizuje na další položku v zásobníku nebo `nil`. Předpokládejte, že bytecode k provedení je uložen v globální proměnné `prog`, která reprezentuje adresový prostor VM, a že první má být provedena instrukce na adrese (indexu) `0`.

```
var
   prog : array[0.. 65535] of Byte;
type
   PReg = ^TReg;
   TReg = record
      value : Longint;
      next : PReg;
   end;
procedure Push(var top : PReg;
               value : Longint);
function Pop(var top : PReg) : Longint;
```

Otázka č. 8

Vysvětlete termín *cloud* a popište jeho výhody a nevýhody ve srovnání s tzv. *cluster*em.

Otázka č. 9

Předpokládejte počítač s plně 16-bitovým procesorem Intel 8086 taktovaným na frekvenci 8 MHz a připojeným na 16-bit variantu 8 MHz sběrnice ISA (20-bit adresový prostor, přenos jednoho slova za 6 taktů). Procesor má 6 bytovou prefetch queue, a výpočetní jednotka procesoru (ALU) pracuje zcela paralelně s načítáním do prefetch queue. Všechny jednotky sdílejí jedno rozhraní k systémové sběrnici, a výpočetní jednotka má vyšší prioritu při potřebě přístupu k ní. V dokumentaci procesoru jsme našli následující tabulku o základním časování instrukcí v ALU:

Instruction	Instruction Operands	Clock Cycles
ADD	`reg,reg`	3
ADD	`reg,mem`	9
ADD	`mem,reg`	16
MOV	`reg,imm`	4

K systémové sběrnici je připojena paměť EPROM namapovaná na adresu `6000:FF00`. V řádkovém debuggeru pod MS-DOSem jsme zjistili následující obsah této paměti (cílový je vždy nejlevější operand, operand v závorkách `[x]` znamená čtení/zápis z/na adresy/u x):

```
-u 6000:FF00
6000:FF00 B800A0      MOV     AX,A000
6000:FF03 2E010610FF  ADD     CS:[FF10],AX
6000:FF08 EB08        JMP     FF12
```

Pokud procesor po zapnutí napájení začne zpracovávat instrukce od adresy `6000:FF00`, tak určete čas v µs mezi dokončením 1. a 2. instrukce (tj. doba zpracování 2. instrukce). Hodnotu můžete uvést ve formě zlomku. Zapište (nakreslete) a detailně vysvětlete i postup výpočtu!

Otázka č. 10

Předpokládejte 32-bitový procesor typu ARM s load/store obecnou registrovou architekturou s 16 univerzálními registry `r0` až `r15`. Procesor má všechny typické příznaky, *příznakový registr ale **ne**ovlivňují instrukce přesunu dat*. Instrukční sada má tříadresové aritmetické instrukce. V assembleru pro tento procesor se považuje nejlevější operand za cílový, operand uvozený znakem # je hodnota immediate, operand v hranatých závorkách `[x]` znamená čtení/zápis z/na adresy/u x, operand `[x, y]` znamená čtení/zápis z/na adresy/u dané součtem hodnot x + y. Tak jako běžné procesory mají speciální podmíněnou variantu skoku (dle podmínky se provede skok nebo instrukce NOP), tak procesory ARM mají podmíněnou variantu pro každou instrukci z instrukčního souboru (v assembleru zapsáno příponou `eq` pro podmínku *equals*, `ne` pro *not equals*). Kód následující funkce Sum zapište v Pascalu (její jediný parametr i návratová hodnota se předávají v registru `r0`) – zapište i její kompletní deklaraci a deklarace všech potřebných typů typu **`record`**:

```
Sum:  mov    r1, r0
      mov    r2, #0
Loop: cmp    r1, #0
      moveq  r0, r2
      reteq
      ldr    r3, [r1, #4]
      add    r2, r2, r3
      ldr    r1, [r1]
      b      Loop
```