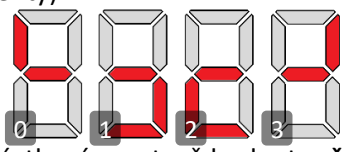
Odpovědi pište na zvláštní odpovědní list s vaším jménem a fotografií. Pokud budete odevzdávat více než jeden list s řešením, tak se na 2. a další listy nezapomeňte podepsat. Do zápatí všech listů vždy napište i/N (kde i je číslo listu, N je celkový počet odevzdaných listů).

Společná část pro otázky označené X

Předpokládejte, že máme k dispozici řadič typu `MAX6958AAPE` od společnosti Maxim sloužící k ovládání zobrazení na 7-segmentových LED displejích. S řadičem se komunikuje pomocí standardní varianty sběrnice I²C. Datasheet tohoto řadiče najdete **v příloze**.

Otázka č. 1 (X)

Předpokládejte, že k řadiči MAX6958 máme připojeny čtyři 7-segmentové LED displeje (nejlevější je připojený jako `digit0`, až nepravější je připojen jako `digit3`) – viz obrázek níže. Na displejích chceme rozsvítit symbol mísy s podstavou, viz obrázek (červená = svítící segmenty, šedá = zhasnuté segmenty):



Napište v šestnáctkové soustavě hodnoty **všech** bytů (bez ACK bitu), které se budou přenášet pro I²C sběrnici v rámci jedné I²C transakce, pokud chceme všechny 4 displeje do cílového stavu rozsvítit právě jedním I²C burst zápisem do řadiče MAX6958.

Otázka č. 2 (X)

V přiloženém datasheetu se v sekci „*Operation with Multiple Masters*“ řeší nějaký problém vznikající v situaci připojení více master zařízení. Má tento problém vztah k tzv. arbitraci I²C sběrnice? Vysvětlete. Jak na I²C sběrnici arbitrace probíhá a co je jejím účelem?

Otázka č. 3 (X)

Předpokládejte, že na hlavní I²C sběrnici jednočipového počítače máme připojený řadič MAX6958, a k tomuto řadiči máme připojen jeden 7-segmentový LED displej jako `digit0`. Dále máme připravenu kostru Pascal programu `afw.pas` firmwaru pro výše uvedený jednočipový počítač:

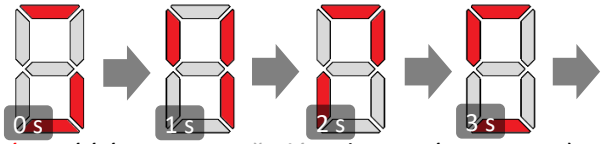
```
program AFW;
uses Crt;
var
  vzor : string;
begin
  vzor := '-.--..';
  { SEM INICIALIZACE }
  while true do begin
    { SEM KROK ANIMACE }
    Delay(1000 { ms } );
  end;
end.
```

a dále máme připravenu proceduru `I2cSend`:

```
type PByte = ^byte;
procedure I2cSend(
  addrByte : byte; data : PByte; count : integer);
```

která odešle `count` bytů po I²C sběrnici, na kterou je řadič displaye připojený. Parametr `addrByte` má standardní format 1. I²C bytu, parametr `data` ukazuje na první z `count` datových bytů, které mají být po I²C sběrnici odeslány. Upravte program tak, aby přečetl obsah proměnné `vzor` a podle toho rozsvítil nebo zhasl jednotlivé segmenty po obvodu „číslice“ – znak tečka (`.`) reprezentuje zhaslý segment, znak pomlčka (`-`) reprezentuje rozsvícený segment. První znak proměnné `vzor` reprezentuje stav horního segmentu, druhý znak stav následujícího segmentu **po** směru hodinových ručiček, atd. Střední segment má být vždy zhasnutý.

Dále program doplňte tak, aby po dobu zapnutí počítače probíhala animace na displeji tak, že vždy po 1 sekundě se stav segmentů o 1 „pootočí“ **proti** směru hodinových ručiček, tj. např. pro uvedenou hodnotu `'-.--..'` v řetězci `vzor` má animace vypadat následujícím způsobem:



(červená = svítící segmenty, šedá = zhasnuté segmenty)

Poznámka: v hlavním programu smíte měnit pouze červeně označené části, nicméně si můžete doprogramovat další procedury a funkce a definovat další globální proměnné.

Otázka č. 4

S využitím ASCII tabulky (**viz příloha**) napište, jaký bude obsah prvních 40 bytů souboru `afw.pas`, který obsahuje zdrojový soubor **ze zadání otázky 3** (hodnotu každého bytu napište **v šestnáctkové** soustavě). Víme, že je text v souboru uložený ve formátu **UTF-16 LE**.

Otázka č. 5

Předpokládejte následující deklarace (kde typ `longword` je 32-bitový celočíselný bezznaménkový, typ `word` je 16-bitový celočíselný bezznaménkový):

```
type PLongword = ^longword; PWord = ^word;
procedure Conv(src : PLongword; dst : PWord);
```

Napište ve Free Pascalu implementaci procedury `Conv` tak, aby převedla vstupní textový null-terminated řetězec `src` z kódování UTF-32 LE do kódování UTF-16 LE a výsledné znaky UTF-16 LE null-terminated řetězce uložila do paměti na místo, kam ukazuje argument `dst` (předpokládejte, že váš kód poběží pouze na little-endian platformách, a že na místě, kam ukazuje proměnná `dst`, je dostatek nevyužité paměti). Unicode znaky z rozsahu `$010000` až `$10FFFF` se v UTF-16 kódují následujícím způsobem:

(1) Od kódu znaku se odečte hodnota `$010000`, a výsledné 20-bitové číslo se rozdělí na dvě 10-bitové části, které se zakódují dle následujících pravidel.

(2) Nejvyšších 10-bitů 20-bitové hodnoty se uloží do nejnižších 10 bitů prvního 16-bitového surrogate znaku (leží na nižší adrese). Horních 6 bitů první surrogate má být nastaveno na (vlevo je hodnota bitu 15, vpravo bitu 10):
`1101 10`

(3) Nejnižších 10-bitů 20-bitové hodnoty se uloží do nejnižších 10 bitů druhého 16-bitového surrogate znaku (leží na vyšší adrese). Horních 6 bitů druhé surrogate má být nastaveno na (od první surrogate se liší pouze 10. bitem):
`1101 11`

Příklad vstupu a správného výstupu procedury `Conv`:

```
src: $00000041 $00002550 $0000FE8D $0001F0B1 $00000042
dst: $0041 $2550 $FE8D $D83C $DCB1 $0042
```

Společná část pro otázky označené Y

Předpokládejte níže popsaný CPU vycházející z architektury procesorů Intel 80386 – je to **32-bitový little-endian** CPU s obecnou registrovou architekturou, s podporou stránkování a s 32-bitovým virtuálním i fyzickým adresovým prostorem. Procesor má obecné registry `EAX`, `EBX`, `ECX`, `EDX`, `DSI`, `EDI`, `EBP`, 32-bitový příznakový registr `EFLAGS` s běžnými příznaky, registr `ESP` (stack pointer, ukazuje na poslední využitý byte, roste dolů), a registr `EIP` (instruction pointer). V instrukční sadě jsou mimo jiné **i následující instrukce** (příznakový registr modifikují pouze aritmetické operace, ale instrukce přenosu dat nikoliv):

- `MOV reg,DWORD PTR imm32/[addr]` (load register)
- `MOV DWORD PTR [addr],reg` (store register)
- `MOV DWORD PTR [addr],imm32` (store constant)
- `MOV reg0,reg1` (transfer from `reg1` to `reg0`)
- `ADD reg,imm32/[addr]/reg` (add without carry)
- `SHL reg,imm32/[addr]/reg` (logical shift left)
- `PUSH imm16/imm32/[addr]/reg`, `POP [addr]/reg`
- `JMP addr` (direct jump), `JZ addr` (jump if zero)
- `CMP DWORD PTR [addr],imm32` (32-bit compare)
- `CALL addr` (direct call)
- `RET` (return from subroutine)

Všechny výše uvedené instrukce se dvěma operandy mají vždy **vlevo cílový** a **vpravo zdrojový** operand. Instrukce mohu mít jednu z následujících variant operandů (povolené varianty viz definice konkrétní instrukce):

- 32-bitový immediate `imm32`
- absolutní adresa `[addr]`, kde `[addr]` může být:
  - `[reg +/- imm]` adresa daná součtem/rozdílem obsahu registru `reg` a konstanty `imm`
- libovolný registr `reg`

Předpokládejte následující kus kódu zapsaného v assembleru tohoto CPU:

```
LABEL1:
  push  ebp
  mov   ebp,esp
  mov   eax,DWORD PTR [ebp+0x8]
  cmp   DWORD PTR [eax],0x0
  jz    LABEL2
  mov   eax,DWORD PTR [ebp+0x8]
  cmp   DWORD PTR [eax],0xfffffffd
  jz    LABEL2
  mov   eax,DWORD PTR [ebp+0x8]
  mov   eax,DWORD PTR [eax]
  shl   eax,1
  mov   edx,DWORD PTR [ebp+0xc]
  add   edx,eax
  mov   eax,DWORD PTR [ebp+0x8]
  mov   DWORD PTR [eax],edx
  push  DWORD PTR [ebp+0xc]
  push  DWORD PTR [ebp+0x10]
  mov   eax,DWORD PTR [ebp+0x8]
  add   eax,0x4
  push  eax
  call  LABEL1
  add   esp,0xc
LABEL2:
  mov   esp,ebp
  pop   ebp
  ret
```

Otázka č. 6 (Y)

Napište v Pascalu bez použití inline assembleru kód procedury (i s deklarací), která by mohla být běžným překladačem přeložena do výše uvedeného kódu v assembleru 80386 (předpokládejte, že procedura používá běžnou Cčkovou volací konvenci, tj. argumenty se předávají na volacím zásobníku zprava doleva, a **odebírá je volající**).

Otázka č. 7 (Y)

Dokážeme z uvedeného strojového kódu (assembleru) poznat, zda byla původní Pascal procedura deklarována s parametry typu `longword` (32-bitový **bez**znaménkový integer), resp. `^longword`, nebo zda byly parametry definovány jako `longint` (32-bitový znaménkový integer), resp. `^longint`? Detailně vysvětlete, jakým způsobem je možné to poznat, resp. proč to poznat není možné.

Otázka č. 8 (Y)

V uvedeném programu je v assembleru zapsána instrukce:

```
cmp    DWORD PTR [eax], 0xfffffffd
```

o které víme, že na úrovni strojového kódu je reprezentována následujícími 3 byty:

```
83 38 fd
```

kde z dokumentace instrukční sady víme, že byte `83` je opcode instrukce `CMP`, a byte `38` pro tuto instrukci říká, že se jedná o 32-bitovou variantu instrukce, a že její levý argument je 32-bit paměťový operand uložený na absolutní adrese dané obsahem registru `EAX`, a pravý operand je immediate hodnota. Vysvětlete, jak je asi definován na úrovni strojového kódu pravý immediate operand této instrukce. Z varianty instrukce je zřejmé, že pravý operand musí být nakonec 32-bitová immediate hodnota, nicméně, jak procesor „ví“, že má 3. byte instrukce s hodnotou `$FD` chápat zrovna jako hodnotu `$FFFFFFFD`? Jaké asi používá „pravidlo“? Detailně vysvětlete.

Otázka č. 9

Na obrázku A níže je vyfocena část PCI Express grafické karty, a na obrázku B část PCI Express řadiče USB sběrnice:

Detailně vysvětlete, proč jsou PCIe konektory obou karet různě dlouhé (s rozdílným počtem pinů), a jaký je mezi nimi rozdíl. Bylo by možné kratší kartu USB řadiče zasunout do dlouhého PCIe slotu na základní desce, kam pasuje grafická karta z obrázku A? Vysvětlete proč, a případně jaké implikace to má.

Otázka č. 10

Vysvětlete, jakým způsobem se typicky ukládají barevná obrazová data, a jaké jsou nejběžnější formáty pixelu (uveďte alespoň 2 různé příklady lišící se kvalitou/přesností reprodukce uloženého obrazu). Pro každý z uvedených formátů pixelů definujte, jaký je význam jednotlivých bytů na offsetech od bázové adresy pixelu.