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).

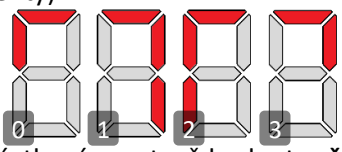
### Společná část pro otázky označené X

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 deštníku 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)

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.
```

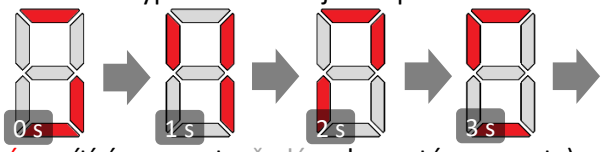
a dále máme připravenu proceduru `I2cSend`:

```
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. **Kód napište co nejefektivněji s vhodným využitím bitových operací (**tedy *krok animace* by již **neměl** pracovat přímo s řetězcem `vzor`, ale již jen s efektivnější reprezentací stavu segmentů**).**

---

### Otázka č. 3

V kontextu nějakého moderního desktopového OS (jako Windows nebo Linux) detailně vysvětlete, co je tzv. *proces* a co tzv. *vlákno*. Jaký je mezi nimi rozdíl, resp. vztah?

---

### Otázka č. 4

Předpokládejte, že chceme reálné číslo `-1027,25` uložit do Pascalové proměnné ve standardním formátu double. Typ `double` je 64-bitové floating-point číslo dle standardu IEEE 754, 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 poslední bit, tedy MSb, je znaménkový bit. Nyní program obsahující takovou proměnnou spustíme na počítači s 32-bitovým little-endian CPU, do proměnné uložíme uvedenou hodnotu `-1027,25`, a zjistili jsme, že je proměnná uložená na adrese `0x0F0444D0`. Napište v šestnáctkové soustavě hodnotu každého bytu paměti, ve kterém bude uložená nějaká část proměnné.

---

### Otázka č. 5

Dopište deklaraci (argumenty s typy) a implementaci procedury `Xform` tak, aby prošla předané pole (předané jako 3. argument) o zadané délce (4. argument), a na každý prvek zavolala předanou čtecí funkci (např. `GetPlus1` v uvedeném testovacím programu), a její výsledek uložila do stejného prvku pole pomocí předané zapisovací procedury (např. `SetTimes2`). Tedy níže uvedený testovací program využívající proceduru `Xform` by měl vypsat `10 26 4 12`:

```
procedure Xform(...); begin ... end;

type PLongword = ^longword;
var  data : array[0..3] of longword;

function GetPlus1(a : PLongword) : longword;
begin GetPlus1 := a^ + 1; end;

procedure SetTimes2(a : PLongword; value : longword);
begin a^ := value * 2; end;

begin
  d[0] := 4; d[1] := 12; d[2] := 1; d[3] := 5;
    Xform(@GetPlus1, @SetTimes2, @d, 4);
  for i := 0 to 3 do Write(d[i], ' ');
end.
```

### Otázka č. 6

Vysvětlete, co je tzv. *zásobníkový stroj* (*stack machine*), a jak se liší od obecné registrové architektury. Bude zásobníkový stroj využívat také koncept volacího zásobníku (call stack)? Vysvětlete proč, a zda/jak volací zásobník případně souvisí se slovem „zásobník“ z názvu stroje.

---

### Otázka č. 7

Předpokládejte následující záznam:

```
type
  TFieldDescriptor = record
    Name : array[0..7] of char;
    Type : byte;
    Offset : longword;
    Length : byte;
    DecimalPlaces : byte;
    Flags : byte;
    AutoincrementNext : longword;
    AutoincrementStep : byte;
  end;
```

Pro každou jeho položku uveďte, na jakém offsetu od začátku záznamu bude ležet (při překladu typickým Pascal překladačem). Je to jediná možnost, jak bude záznam v paměti vypadat, nebo je možné nějak Pascalový překladač přesvědčit i k jiné paměťové organizaci? Vše detailně vysvětlete.

---

### 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)
- `ADC reg,imm32/[addr]/reg` (add with carry)
- `SUB reg,imm32/[addr]/reg` (subtract without carry)
- `DIV reg,imm32/[addr]/reg` (divide left arg. by right arg.)
- `PUSH imm16/imm32/[addr]/reg`, `POP [addr]/reg`
- `JMP addr` (direct jump), `JNE addr` (jump if not equal)
- `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`

---

### Otázka č. 8 (Y)

Předpokládejte následující kus kódu zapsaného v assembleru tohoto CPU:

```
LABEL1:
  push  ebp
  mov   ebp,esp
  sub   esp,0x4
  mov   eax,DWORD PTR [ebp+0x8]
  cmp   DWORD PTR [eax],0x0
  jne   LABEL2
  mov   eax,DWORD PTR [ebp+0xc]
  div   eax,DWORD PTR [ebp+0x10]
  mov   DWORD PTR [ebp-0x4],eax
  jmp   LABEL3
LABEL2:
  mov   eax,DWORD PTR [ebp+0x10]
  add   eax,0x1
  push  eax
  mov   eax,DWORD PTR [ebp+0x8]
  mov   edx,DWORD PTR [eax]
  mov   eax,DWORD PTR [ebp+0xc]
  add   eax,edx
  push  eax
  mov   eax,DWORD PTR [ebp+0x8]
  add   eax,0x4
  push  eax
  call  LABEL1
  add   esp,0xc
  mov   DWORD PTR [ebp-0x4],eax
LABEL3:
  mov   eax,DWORD PTR [ebp-0x4]
  mov   esp,ebp
  pop   ebp
  ret
```

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

---

### Otázka č. 9 (Y)

Předpokládejte, že by uvedený procesor neměl instrukci celočíselného dělení, a přesto bychom program z otázky 8, chtěli přeložit Pascal překladačem pro tento cílový procesor. Detailně vysvětlete, jak by si s tím typický Pascal překladač nejspíše poradil, a zda, resp. jak by se asi změnil vygenerovaný strojový kód (napište co nejpřesněji na úrovni assembleru).

---

### Otázka č. 10 (Y)

Zapište v assembleru uvedeného CPU, jak by asi typický Pascal překladač přeložil níže uvedenou proceduru `Add` (typ `qword` je 64-bitový bezznaménkový integer). Předpokládejte, že proměnná `a` bude ležet na adrese `0x40c000`, proměnná `b` na adrese `0x40c008`.

```
var
  a, b : qword;
procedure Add(v : qword); cdecl;
begin
  a := b + v + 3;
end;
```