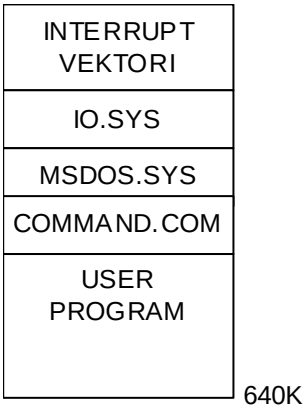
Mehanizam prekida

Struktura OS

DOS:



Teorija

Pojmovi

- Adresa prekidne rutine, ili kraće prekida (*interrupt pointer*)
- Tabela adresa prekidnih rutina (*Interrupt vector table*)
- Dozvola i zabrana izvršenja prekida: *Interrupt flag* i *Trap flag* u PSW (*Program Status Word*) registru

Kod 8086ice kada se dogodi prekid, **hardver** izvršava sledeću proceduru:

- Određuje se broj prekida N
- Registri PSW, CS i IP se stavljaju na stek u tom redosledu
- Resetuju se bitovi IF i TF iz registra PSW
- Stavlja se sadržaj memorijske lokacije 4*N u registar IP i sadržaj lokacije 4*N+2 u registar CS (*interrupt pointer* se smešta u PC)

Da bi se izvršio povratak iz prekidne rutine, potrebno je izvršiti (softverski) naredbu **IRET**, koja će skinuti sa steka registre koje je hardver sačuvao prilikom obrade prekida.

Zad 0. Napisati program na C jeziku koji broji događaje koje je generisala tastatura. Broj prekida na koji je zakačena tastatura je 9h. Adresa registra podataka tastature je 60h, a kontrolnog registra je 61h. Adresa kontrolnog registra kontrolera periferije je 20h.

```
#include <stdio.h>

volatile int cntr = 0;

//interrupt kljucna rec oznacava da se radi o prekidnoj
//rutini (cuva na steku i restaurira registre AX,... i na
//kraju poziva IRET)
void interrupt keyboardISR(){
     asm {
         in      al, 60h         //; Get scan code
     }

                                 //;
    ++cntr;                      //; Process event
                                 //;

    //; Protokol završetak obrade prekida.
    //; Obaveštava se periferija kontroler prekida da je
    //; prekid obrađen
    asm {
        in      al, 61h  //; Send acknowledgment without
        or      al, 10000000b   //;   modifying the other
                                //;    bits.
        out     61h, al         //;
        and     al, 01111111b   //;
        out     61h, al         //;
        mov     al, 20h         //; Send End-of-Interrupt
                                //; signal to the Interrupt
        out     20h, al         //; controller
     }
}

unsigned oldKbdOFF, oldKbdSEG; // stara prekidna rutina

// postavlja novu prekidnu rutinu u Interrupt vektor
// tabeli
void inic(){
  asm{
        cli         // Zabranjuju se prekidi
        push es
        push ax

        mov ax,0    // inicijalizuje rutinu za tajmer
```

```
        mov es,ax  // es = 0

        // pamti staru rutinu
        // oldKbdSEG = [00026h]; - 9*4h+2h=26h
        mov ax, word ptr es:0026h
        mov word ptr oldKbdSEG, ax
        // oldKbdOFF = [00024h]; - 9*4h=24h
        mov ax, word ptr es:0024h
        mov word ptr oldKbdOFF, ax

        // postavlja novu rutinu
        mov word ptr es:0026h, seg keyboardISR
        mov word ptr es:0024h, offset keyboardISR

        pop ax
        pop es
        sti       // Dozvoljavaju se prekidi
  }
}

// restauira staru prekidnu rutinu u Interrupt vektor
// tabeli
void restore(){
  asm {
          cli
          push es
          push ax

          mov ax,0
          mov es,ax


          // vraća originalnu rutinu rutinu
          mov ax, word ptr oldKbdSEG
          mov word ptr es:0026h, ax
          mov ax, word ptr oldKbdOFF
          mov word ptr es:0024h, ax

          pop ax
          pop es
          sti
  }
}
```

```
void doSomething(){
  asm sti

  for (int i = 0; i < 10; ++i) {
     printf("main %d\n", i);

     for (int j = 0; j< 30000; ++j)
          for (int k = 0; k < 30000; ++k);
  }

  printf("Tastatura je generisala %d dogadjaja!\n", cntr);
  // Napomena: periferija generise prekid pri pritiskanju i
  // otpustanju tastera

}

int main(){
  inic();

  doSomething();

  restore();

  return 0;
}
```

Zad 1. Napisati program na C jeziku koji prekida izvršenje glavnog programa na svakih 5s i ispisuje na ekran odgovarajuću poruku. Prekid od *timer-a* se generiše 18.2 puta u sekundi. Ovaj prekid se nalazi na broju 1Ch.

Rešenje:

```
#include <iostream.h>

volatile int brojac = 100;

// prekidna rutina
void interrupt timer(){
     if (--brojac==0) {
          brojac = 100;
          cout << "............................" << endl;
     }
}

unsigned oldTimerOFF, oldTimerSEG; // stara prekidna rutina
```

```
// postavlja novu prekidnu rutinu
void inic(){
  asm{
        cli
        push es
        push ax

        mov ax,0   // inicijalizuje rutinu za tajmer
        mov es,ax

         // pamti staru rutinu
         // oldTimerSEG = [00072h];
         mov ax, word ptr es:0072h
         mov word ptr oldTimerSEG, ax
         // oldTimerOFF = [00070h];
         mov ax, word ptr es:0070h
         mov word ptr oldTimerOFF, ax


        mov word ptr es:0072h, seg timer //;  postavlja moju
                                         // rutinu
        mov word ptr es:0070h, offset timer

        pop ax
        pop es
        sti
  }
}

// vraca staru prekidnu rutinu
void restore(){
  asm {
          cli
          push es
          push ax

          mov ax,0
          mov es,ax


          mov ax, word ptr oldTimerSEG
          mov word ptr es:0072h, ax
          mov ax, word ptr oldTimerOFF
          mov word ptr es:0070h, ax

          pop ax
          pop es
```

```
          sti
  }
}

// Zabranjuje prekide
#define lock asm cli

// Dozvoljava prekide
#define unlock asm sti


void doSomething(){
  for (int i = 0; i < 30; ++i) {

     // Ispis ne sme da se prekida!!
     // konzola deljeni resurs
     lock
     cout << i << endl;
     unlock

     for (int j = 0; j< 30000; ++j)
          for (int k = 0; k < 30000; ++k);
  }

}

int main(){
  inic();

  doSomething();

  restore();

  return 0;
}
```

Zad2. Napisati program kao u zadatku 1, ali da se iz glavnog programa kontrolise da li su prekidi dozvoljeni ili ne.

Rešenje:

```
#include <iostream.h>

volatile int brojac = 100;

// prekidna rutina
```

```
void interrupt timer(){
        if (--brojac==0) {
          brojac = 100;
          cout << "........................." << endl;
     }
}

unsigned oldTimerOFF, oldTimerSEG; // stara prekidna rutina

// postavlja novu prekidnu rutinu
void inic(){
  asm{
        cli
        push es
        push ax

        mov ax,0   // inicijalizuje rutinu za tajmer
        mov es,ax

        // pamti staru rutinu
        // oldTimerSEG = [00072h];
        mov ax, word ptr es:0072h
        mov word ptr oldTimerSEG, ax
        // oldTimerOFF = [00070h];
        mov ax, word ptr es:0070h
        mov word ptr oldTimerOFF, ax


        mov word ptr es:0072h, seg timer // postavlja moju
                                         // rutinu
        mov word ptr es:0070h, offset timer  //

        pop ax
        pop es
        sti
  }
}


// vraca staru prekidnu rutinu
void restore(){
  asm {
          cli
          push es
          push ax

          mov ax,0
```

```
            mov es,ax


            mov ax, word ptr oldTimerSEG
            mov word ptr es:0072h, ax
            mov ax, word ptr oldTimerOFF
            mov word ptr es:0070h, ax

            pop ax
            pop es
            sti
  }
}

//Zabranjuje prekide - na ovaj nacin omoguceno je
//ugnezdavanje parova poziva lock-unlock
#define lock asm{\
            pushf;\
            cli;\
            }

// Dozvoljava prekide
#define unlock asm popf

void doSomething(){
  lock
  for (int i = 0; i < 30; ++i) {

     lock
     cout << i << endl;
     unlock

     if (i == 7) unlock
     else if (i == 25) lock

     for (int j = 0; j< 30000; ++j)
          for (int k = 0; k < 30000; ++k);
  }
  unlock
}

int main(){
  inic();
  doSomething();
  restore();
  return 0;
}
```

Zad 3. Realizovati promenu konteksta (engl. context switch) koristeći prekid od timera (08h). Kontekst procesora čuvati na steku. Predvideti da se svakoj niti u sistemu može dodeliti različit kvant vremena za izvršavanje. Demonstrirati rad sistema ako se u njemu nalaze dve niti, od kojih je jednoj dodeljen duplo veći kvant vremena nego drugoj.

```
//pretpostavljeni memorijski model: huge

#include <iostream.h>
#include <dos.h>

// Zabranjuje prekide
#define lock asm cli

// Dozvoljava prekide
#define unlock asm sti

struct PCB{
     unsigned sp;
     unsigned ss;
     unsigned zavrsio;
     int kvant;
};

PCB *p[3];
volatile PCB* running;

volatile int nextThread = 2;
PCB* getNextPCBToExecute(){
     if (nextThread == 1)
          nextThread = 2;
     else nextThread = 1;
     if (p[nextThread]->zavrsio){
          if (nextThread == 1)
               nextThread = 2;
          else nextThread = 1;
          if (p[nextThread]->zavrsio)
               nextThread = 0;
     }
     return p[nextThread];
}


//pomocne promenljive za prekid tajmera
unsigned tsp;
unsigned tss;
```

```
volatile int brojac = 20;
volatile int zahtevana_promena_konteksta = 0;

void interrupt timer(){  // prekidna rutina
     if (!zahtevana_promena_konteksta) brojac--;
     if (brojac == 0 || zahtevana_promena_konteksta) {
          asm {
               // cuva sp
               mov tsp, sp
               mov tss, ss
          }

          running->sp = tsp;
          running->ss = tss;

          running= getNextPCBToExecute();    // Scheduler

          tsp = running->sp;
          tss = running->ss;

          brojac = running->kvant;

          asm {
               mov sp, tsp   // restore sp
               mov ss, tss
          }
     }

     // poziv stare prekidne rutine koja se
     // nalazila na 08h, a sad je na 60h
     // poziva se samo kada nije zahtevana promena
     // konteksta - tako se da se stara
     // rutina poziva samo kada je stvarno doslo do prekida
     if(!zahtevana_promena_konteksta) asm int 60h;

     zahtevana_promena_konteksta = 0;
}

void dispatch(){ // sinhrona promena konteksta
     asm cli;
     zahtevana_promena_konteksta = 1;
     timer();
     asm sti;
}

unsigned oldTimerOFF, oldTimerSEG; // stara prekidna rutina
```

```
// postavlja novu prekidnu rutinu
void inic(){
     asm{
          cli
          push es
          push ax

          mov ax,0   //  ; inicijalizuje rutinu za tajmer
          mov es,ax

          mov ax, word ptr es:0022h //; pamti staru rutinu
          mov word ptr oldTimerSEG, ax
          mov ax, word ptr es:0020h
          mov word ptr oldTimerOFF, ax

          mov word ptr es:0022h, seg timer    //postavlja
          mov word ptr es:0020h, offset timer //novu rutinu

          mov ax, oldTimerSEG  //  postavlja staru rutinu
          mov word ptr es:0182h, ax //; na int 60h
          mov ax, oldTimerOFF
          mov word ptr es:0180h, ax

          pop ax
          pop es
          sti
     }
}

// vraca staru prekidnu rutinu
void restore(){
     asm {
          cli
          push es
          push ax

          mov ax,0
          mov es,ax


          mov ax, word ptr oldTimerSEG
          mov word ptr es:0022h, ax
          mov ax, word ptr oldTimerOFF
          mov word ptr es:0020h, ax

          pop ax
          pop es
```

```
            sti
        }
}

void exitThread(){
        running->zavrsio = 1;
        dispatch();
}

void a(){
        for (int i =0; i < 30; ++i) {
            lock
            cout<<"u a() i = "<<i<<endl;
            unlock
            for (int k = 0; k<10000; ++k)
                for (int j = 0; j <30000; ++j);
        }
        exitThread();
}

void b(){
        for (int i =0; i < 30; ++i) {
            lock
            cout<<"u b() i = "<<i<<endl;
            unlock
            for (int k = 0; k<10000; ++k)
                for (int j = 0; j <30000; ++j);
        }
        exitThread();
}


void createProcess(PCB *newPCB, void (*body)()){
        unsigned* st1 = new unsigned[1024];

        st1[1023] =0x200;//setovan I fleg u
                         // pocetnom PSW-u za nit
        st1[1022] = FP_SEG(body);
        st1[1021] = FP_OFF(body);

        newPCB->sp = FP_OFF(st1+1012); //svi sacuvani registri
                                       //pri ulasku u interrupt
                                       //rutinu
        newPCB->ss = FP_SEG(st1+1012);
        newPCB->zavrsio = 0;
}
```

```
void doSomething(){
     lock
     p[1] = new PCB();
     createProcess(p[1],a);
     cout<<"napravio a"<<endl;
     p[1]->kvant = 40;

     p[2] = new PCB();
     createProcess(p[2],b);
     cout<<"napravio b"<<endl;
     p[2]->kvant = 20;

     p[0] = new PCB();

     running = p[0];
     unlock

     for (int i = 0; i < 30; ++i) {
     lock
     cout<<"main "<<i<<endl;
     unlock

     for (int j = 0; j< 30000; ++j)
          for (int k = 0; k < 30000; ++k);
     }
     cout<<"Happy End"<<endl;
}

int main(){

  inic();

  doSomething();

  restore();

  return 0;
}
```

Zad 4. Prokomentarisati prednosti pristupa u kome je semantika kreiranja niti analogna onoj na jeziku Java: nit je aktivan objekat klase izvedene iz klase Thread koju definiše korisnik, u odnosu na tradicionalan pristup gde se nit kreira nad nekom globalnom funkcijom. Takođe, objasniti u čemu bi bio problem ukoliko bi se kreirana nit implicitno pokretala odmah po kreiranju unutar konstruktora osnovne klase Thread.

Rešenje:

- U tradicionalnom pristupu nit se kreira nad nekom globalnom funkcijom programa.
  - ovaj pristup nije dovoljno fleksibilan
    - često je beskorisno kreirati više niti nad istom funkcijom ako one ne mogu da se međusobno razlikuju, npr. pomoću argumenata pozvane funkcije.
  - U tradicionalnim sistemima se često omogućuje da korisnička funkcija nad kojom se kreira nit dobije neki argument prilikom kreiranja niti.
    - Mana - broj i tipovi ovih argumenata su fiksni, definisanim samim sistemom
- Bolji pristup - nit je predstavljena klasom `Thread`, i može se definisati kao aktivan objekat, koji poseduje sopstveni tok kontrole (sopstveni stek poziva). Svi ugnežđeni pozivi, zajedno sa svojim automatskim objektima, odvijaju u sopstvenom kontekstu te niti.
- Nit je objekat klase izvedene iz klase `Thread` koju definiše korisnik. Korisnik kreira nit kreiranjem objekta ove klase.
- Nit se kreira nad polimorfnom operacijom `run()` klase `Thread` koju korisnik može da redefiniše u izvedenoj klasi.
- Svaki aktivni objekat iste klase poseduje sopstvene atribute - na taj način se razlikuju aktivni objekti iste klase (niti nad istom funkcijom)
- Jedini (skriveni) argument funkcije `run()` nad kojom se kreira nit jeste pokazivač ***this***, koji ukazuje na čitavu strukturu proizvoljnih atributa objekta

```
class Thread {
public:

  Thread ();
  Thread (void (*body)());  //kreiranje niti nad globalnom funkcijom
  void start ();

protected:

  virtual void run() {} //polimorfna metoda

};
```

- Funkcija `start()` služi za eksplicitno pokretanje niti.
- Implicitno pokretanje moglo je da se obezbedi tako što se nit pokreće odmah po kreiranju, što bi se realizovalo unutar konstruktora osnovne klase `Thread`.
- Problem:
  - Konstruktor osnovne klase izvršava pre konstruktora izvedene klase

- može se dogoditi da novokreirana nit počne izvršavanje pre nego što je kompletan objekat izvedene klase kreiran.

- Kako nit izvršava redefinisanu funkciju `run()`, a unutar ove funkcije može da se pristupa članovima, moglo bi da dođe do konflikta.

Zad 5. Dat je sledeći kod:

```
class Calculate : public Thread {
public:
  virtual void run () {
    // A long calculation ...
  }
  ...
}

Calculate* aCalculation = new Calculate();
```

Objasniti u čemu je razlika između:

```
aCalculation->run();
```

i

```
aCalculation->start();
```

Zad 6. – Prvi kolokvijum Mart 2010.
Na narednoj strani prikazan je najvažniji deo programa koji pronalazi put kroz lavirint (engl. *maze*) poput onog datog na slici dole. Objašnjenje nekih pomoćnih operacija je sledeće:

- `Direction operator-(Position pos2, Position pos1)`: Vraća smer u kome se polje označeno pozicijom `pos2` u kvadratnoj koordinatnoj mreži može dostići iz jednog od četiri susedna polja na poziciji `pos1`. Na primer, ako je `pos2` desni sused od `pos1`, vratiće E; ako mu je donji sused, vratiće S; ako mu je levi, vratiće W; ako je gornji, vratiće N.
- `int Maze::isExit(Position)`: Vraća 1 ako je polje na datoj poziciji u lavirintu izlaz, odnosno 0 ako nije.
- `Position* Maze::getFirstOption(Position p, Direction d)`, `getSecondOption, getThirdOption`: Za dato polje `p` u lavirintu u koje se stiglo napredovanjem u datom smeru `d`, vraćaju slobodna susedna polja u koja se može ići bez povratka u polje iz koga se stiglo. Ako nema ni jednog mogućeg smera (slobodnog susednog polja osim onoga iz koga se stiglo u `p`), sve tri operacije vratiće 0; ako je moguće nastaviti samo u jednom smeru, samo će `getFirstOption` vratiti poziciju susednog slobodnog polja, ostale će vratiti 0; ako je moguće nastaviti u dva smera, prve dve operacije će vratiti pozicije susednih slobodnih polja, treća će vratiti 0.

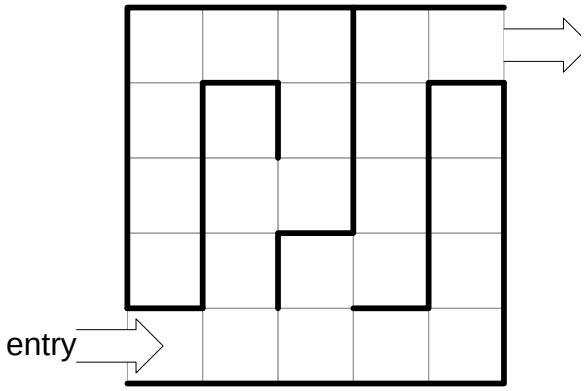
Tragač za izlazom pokrenut je na sledeći način za lavirint prikazan na donjoj slici:

```
MazeExitSeeker* seeker = new MazeExitSeeker(theMaze,entry,E);
seeker->start();
```

a)(4) Precizno objasniti kako ovaj program pretražuje lavirint.

b)(3) Koliko ukupno niti će biti pokrenuto za ovaj lavirint, računajući i početnu nit (`seeker`) prikazanu gore?

c)(3) Uz pretpostavku da svaka nit ima svoj logički izlazni uređaj `cout` (nema preplitanja ispisa iz različitih niti), koliko puta će ukupno biti ispisana rečenica „I have reached a dead end and I am giving up.”?




```
enum Direction {N, E, S, W};

class Position {
public:
  friend Direction operator- (Position pos2, Position pos1);
  ...
};

class Maze {
public:
  int isExit(Position);
  Position* getFirstOption(Position,Direction);
  Position* getSecondOption(Position,Direction);
  Position* getThirdOption(Position,Direction);
  ...
};

class MazeExitSeeker : public Thread {
public:
  MazeExitSeeker (Maze* m, Position start, Direction dir) :
      myMaze(m), myPos(start), myDir(dir) {}
protected:
  virtual void run();
private:
  Maze* myMaze;
  Position myPos;
  Direction myDir;
};

void MazeExitSeeker::run () {
  cout<<”I’m starting my search from “<<myPos<<” towards “<<myDir<<.\n”
  while (1) {
    if (myMaze->isExit(myPos)) {
      cout<<”I have found the exit!\n“
```

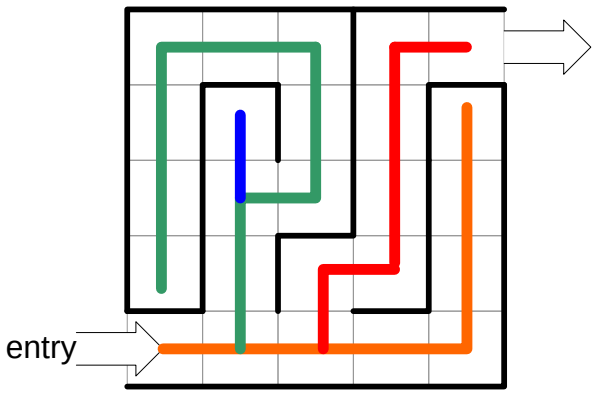
```
      return;
    }
    Position* nextPos1 = myMaze->getFirstOption(myPos,myDir);
    Position* nextPos2 = myMaze->getSecondOption(myPos,myDir);
    Position* nextPos3 = myMaze->getThirdOption(myPos,myDir);
    if (nextPos1==0) {
      cout<<"I have reached a dead end and I am giving up.\n"
      return;
    }
    if (nextPos2!=0)
      (new MazeExitSeeker(myMaze,*nextPos2,*nextPos2-myPos))->start();
    if (nextPos3!=0)
      (new MazeExitSeeker(myMaze,*nextPos3,*nextPos3-myPos))->start();
    myDir=*nextPos1-myPos;
    myPos=*nextPos1;
  }
}
```

Rešenje:

a) Svaka nit napreduje kroz hodnike lavirinta i u svakom koraku, na datom polju u kvadratnom koordinatnom sistemu, radi sledeće. Najpre pogleda da li je pronašla izlaz i ako jeste, ispisuje to i završava se. Zatim od lavirinta dobije najviše tri susedna slobodna polja u koja može da pređe (osim onoga iz koga je stigla). Ako ni jedno od takvih ne postoji, ta nit je udarila u slepi hodnik, pa ispisuje rečenicu „I have reached a dead end and I am giving up“ i gasi se. Inače, ta nit nastavlja da ide prvom dobijenom opcijom, tako da joj je sledeće polje to susedno polje dobijeno kao prva opcija, u tom smeru. Ako postoje druga i treća opcija, onda ova nit kreira nove niti, po jednu za svaku od tih raspoloživih opcija (drugu i treću), tako da te niti nastavljaju od tih susednih polja, u odgovarajućim smerovima.

b)(3) Četiri niti. Na slici je prikazan jedan mogući raspored puteva kojim tragaju te niti, uz pretpostavku da roditeljska nit nastavlja desnim od mogućih puteva, a potomci ostalim.

c)(3) Tri puta.



Zad 7. – Prvi kolokvijum Mart 2011.
Date su deklaracije pokazivača preko kojih se može pristupiti registrima dva izlazna uređaja:

```
typedef unsigned int REG;
REG* io1Ctrl =...;   // Device 1 control register
REG* io1Status =...; // Device 1 status register
REG* io1Data =...;   // Device 1 data register
REG* io2Ctrl =...;   // Device 2 control register
REG* io2Status =...; // Device 2 status register
REG* io2Data =...;   // Device 2 data register
```

U upravljačkim registrima najniži bit je bit *Start* kojim se pokreće uređaj, a u statusnim registrima najniži bit je bit spremnosti (*Ready*). Svi registri su veličine jedne mašinske reči (tip `unsigned int`). Biti spremnosti statusnih registara oba uređaja vezani su preko logičkog ILI kola na isti signal zahteva za prekid, tako da se isti prekid generiše ako je bilo koji (ili oba) uređaja postavio svoj bit spremnosti.

Potrebno je napisati kod koji može da izvrši prenos datog bloka podataka korišćenjem oba uređaja uporedo, tako da se na svaki uređaj prenese po pola datog bloka (ili približno pola, ako je broj reči neparan). Ovaj prenos pokreće se pozivom dole date funkcije `transfer()`.

```
int flag1 = 0, flag2 = 0; // I/O completed
unsigned int index1, count1;
unsigned int index2, count2;
REG* buf1;
REG* buf2;

void transfer (REG* buffer, unsigned int count) {
  count1 = count/2;
  count2 = count1+count%2;
  buf1 = buffer;
  buf2 = buffer+count1;
  index1 = index2 = 0;
  flag1 = flag2 = 0;
  *io1Ctrl = 1;
  *io1Ctr2 = 1;
}

int isIOCompleted () {
  return flag1 && flag2;
}

interrupt void ioInterrupt();
```

Napisati kod prekidne rutine `ioInterrupt()`.

Rešenje:

```
interrupt void ioInterrupt() {
  if (*ioStatus1) {
    *io1Data = buf1[index1++];
```

```
    if (--count1 == 0) {
      flag1 = 1;
      *iosCtrl1 = 0;
    }
  }
  if (*ioStatus2) {
    *io2Data = buf2[index2++];
    if (--count2 == 0) {
      flag2 = 1;
      *iosCtrl2 = 0;
    }
  }
}
```