

# Prova pratica di Calcolatori Elettronici

*C.d.L. in Ingegneria Informatica, Ordinamento DM 270*

8 giugno 2022

1. Siano date le seguenti dichiarazioni, contenute nel file `cc.h`:

```
struct st1 {
    char vc[2];
};
class cl {
    st1 s;
    long v[2];
public:
    cl(char c, st1& s2);
    void elab1(st1& s1);
    void stampa()
    {
        for (int i = 0; i < 2 ;i++) cout << s.vc[i] << ' '; cout << endl;
        for (int i = 0; i < 2; i++) cout << v[i] << ' '; cout << endl << endl;
    }
};
```

Realizzare in Assembler GCC le funzioni membro seguenti.

```
cl::cl(char c, st1& s2)
{
    for (int i = 0; i < 2; i++) {
        s.vc[i] = c;
        v[i] = s2.vc[i] - c;
    }
}
void cl::elab1(st1& s1)
{
    cl cla('p', s1);
    for (int i = 0; i < 2; i++) {
        if (s.vc[i] < s1.vc[i]) {
            s.vc[i] = cla.s.vc[i];
            v[i] = cla.v[i] + i;
        }
    }
}
```

2. Colleghiamo al sistema delle periferiche PCI di tipo `ce`, con `vendorID 0xedce` e `deviceID 0x1234`. Ogni periferica `ce` usa 8 byte nello spazio di I/O a partire dall'indirizzo base specificato nel registro di configurazione BAR0, sia `b`.

La periferiche **ce** sono periferiche di ingresso in grado di generare richieste di interruzione. Ciascuna periferica contiene un certo numero di canali, numerati da 0 a `MAX_CHAN - 1`, ciascuno in grado di operare indipendentemente dagli altri. Questo permette di avere più trasferimenti attivi contemporaneamente. Per attivare un canale è necessario settare il corrispondente bit nel registro di controllo (**CTL**). La periferica contiene un unico registro di ingresso, **RBR**, e un unico piedino per generare richieste di interruzione, condivisi tra tutti canali. Quando un canale attivo produce un nuovo valore e trova **RBR** libero, deposita il valore in **RBR**, scrive il numero del canale nel registro **CHN** e invia una richiesta di interruzione. Il registro **RBR** risulterà poi occupato fino a quando non verrà letto dal software.

I registri accessibili al programmatore sono i seguenti:

1. **CTL** (indirizzo  $b$ , 1 byte, lettura/scrittura): registro di controllo; il bit  $i$ -esimo permette di attivare (1) o disattivare (0) il canale  $i$ -esimo;
2. **CHN** (indirizzo  $b + 4$ , 1 byte, lettura): (**CH**annel **N**umber) se la periferica ha inviato una richiesta di interruzione, il registro contiene il numero del canale che ha prodotto il valore contenuto in **RBR**;
3. **RBR** (indirizzo  $b + 8$ , 1 byte, lettura): registro di ingresso;

L'interfaccia genera una interruzione se uno dei canali attivi ha depositato un nuovo valore in **RBR**. La lettura di **RBR** funge da risposta alla richiesta di interruzione: l'interfaccia non presenta nuovi valori in **RBR** e non cambia il contenuto di **CHN** fino a quando **RBR** non viene letto.

Vogliamo fornire all'utente una primitiva

```
void ceread(natl id, char *buf, natl quanti);
```

Il parametro `id` identifica una delle periferiche **ce** installate. La primitiva permette di leggere da tale periferica una sequenza di `quanti` byte dal primo canale (in ordine di identificatore) non attualmente già attivo. Se tutti i canali sono attivi la primitiva attende che se ne liberi uno. I byte letti saranno scritti a partire dall'indirizzo `buf`.

Per descrivere le periferiche **ce** aggiungiamo le seguenti strutture dati al modulo I/O:

```
static const int MAX_CHAN = 4;
struct des_chan {
    natl sync;
    char *buf;
    natl quanti;
};
struct des_ce {
    ioaddr iCTL, iCHN, iRBR;
    natl mutex;
    natl free_chan;
    des_chan chan[MAX_CHAN];
};
static const int MAX_CE = 16;
des_ce array_ce[MAX_CE];
natl next_ce;
```

I primi `next_ce` elementi del vettore `array_ce` contengono i descrittori, opportunamente inizializzati, delle periferiche di tipo **ce** effettivamente rilevate in fase di avvio del sistema. Ogni periferica è identificata dall'indice del suo descrittore. I descrittori `des_ce` contengono gli indirizzi dei registri della periferica (`iCTL`, `iCHN` e `iRBR`), l'indice di un semaforo `mutex` per l'accesso in mutua esclusione alle risorse condivise tra i canali (in particolare il registro **CTL**), un semaforo `free_chan` che contiene un gettone per ogni canale libero, e un array di descrittori `des_chan`, con un elemento per ogni canale. I descrittori `des_chan` contengono i dati del trasferimento attivo sul canale (`buf` e `quanti`) e un semaforo `sync` su cui si sospende il processo che ha richiesto il trasferimento.

Modificare i file `io.s` e `io.cpp` in modo da realizzare la primitiva come descritto.