Prova pratica di Calcolatori Elettronici

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

6 luglio 2016

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

```
struct st1 { char vi[4]; }; struct st2 { int vd[4]; };
class cl
{       char v1[4]; char v2[4]; long v3[4];
public:
        cl(st1 ss); cl(st1 s1, long ar2[]);
        cl elab1(char ar1[], st2 s2);
        void stampa()
        {       char i;
                for (i=0;i<4;i++) cout << (int)v1[i] << ' '; cout << endl;
                for (i=0;i<4;i++) cout << (int)v2[i] << ' '; cout << endl;
                for (i=0;i<4;i++) cout << v3[i] << ' '; cout << endl << endl;
        }
};
```

Realizzare in Assembler GCC le funzioni membro seguenti.

```
cl::cl(st1 ss)
{
        for (int i = 0; i < 4; i++) {
                v1[i] = v2[i] = ss.vi[i]; v3[i] = ss.vi[i] + ss.vi[i];
        }
}
cl::cl(st1 s1, long ar2[])
{
        for (int i=0; i<4; i++) {
                v1[i] = v2[i] = s1.vi[i]; v3[i] = ar2[i];
        }
}
cl cl::elab1(char ar1[], st2 s2)
{
        st1 s1;
        for (int i = 0; i < 4; i++)
                s1.vi[i] = ar1[i];
        cl cla(s1);
        for (int i = 0; i < 4; i++)
                cla.v3[i] = s2.vd[i];
        return cla;
}
```

2. Colleghiamo al sistema delle periferiche PCI di tipo ce, con vendorID 0xedce e deviceID 0x1234. Ogni periferica ce usa 16 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 interruzioni. I registri accessibili al programmatore sono i seguenti:

   1. **CTL** (indirizzo b, 1 byte): registro di controllo; il bit numero 0 permette di abilitare (1) o disabilitare (0) le richieste di interruzione;
   2. **STS** (indirizzo $b + 4$, 1 byte): registro di stato; il bit numero 0 vale 1 se e solo se il registro RBR contiene un dato non ancora letto;
   3. **RBR** (indirizzo $b + 8$, 1 byte): registro di lettura;

   L'interfaccia genera una interruzione se le interruzioni sono abilitate e il registro RBR contiene un valore non ancora letto. L'interfaccia non presenta nuovi valori in RBR se questo ne contiene uno non ancora letto, quindi la lettura di RBR funge da risposta alla richiesta di interruzione.

   Vogliamo fornire all'utente una primitiva

   ```
   ceread(natl id, char *buf, natl& quanti, char stop)
   ```

   Il parametro id identifica una delle periferiche ce installate. La primitiva permette di leggere da tale periferica una sequenza di byte che termina con il carattere stop passato come quarto argomento. I byte letti saranno scritti a partire dall'indirizzo buf. Il parametro quanti è usato sia come argomento di ingresso che di uscita: in ingresso l'utente specifica il numero massimo di byte da leggere (anche se stop non è stato ricevuto) e in uscita la primitiva dice all'utente il numero di byte che sono stati effettivamente letti (che può essere inferiore al massimo, quando si riceve stop).

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

   ```
   des_ce array_ce[MAX_CE];
   natl next_ce;
   ```

   I primi next_ce elementi del vettore array_ce contengono i destrittori, opportunamente inizializzati, delle periferiche di tipo ce effettivamente rilevate in fase di avvio del sistema. Ogni periferica è identificata dall'indice del suo descrittore. La struttura des_ce deve essere definita dal candidato.

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

```
/**
 * File: cc.h
 *       Contains the declaration for the data structures used in the exercise.
 *
 * Author: Rambod Rahmani <rambodrahmani@autistici.org>
 *         Created on 10/09/2019.
 */

#include <iostream>

using namespace std;

struct st1
{
    char vi[4];
};

struct st2
{
    int vd[4];
};

class cl
{
    char v1[4];
    char v2[4];
    long v3[4];

public:
    cl(st1 ss); cl(st1 s1, long ar2[]);

    cl elab1(char ar1[], st2 s2);

    void stampa()
    {
        char i;

        for (i=0;i<4;i++)
        {
            cout << (int)v1[i] << ' ';
        }

        cout << endl;

        for (i=0;i<4;i++)
        {
            cout << (int)v2[i] << ' ';
        }

        cout << endl;

        for (i=0;i<4;i++)
        {
            cout << v3[i] << ' ';
        }

        cout << endl << endl;
    }
};
```

```
/**
 * File: es1.cpp
 *       Contains the C++ code to be translated into Assembly (es1.s file).
 *
 * Author: Rambod Rahmani <rambodrahmani@autistici.org>
 *         Created on 10/09/2019.
 */

#include "cc.h"

cl::cl(st1 ss)
{
    for (int i = 0; i < 4; i++)
    {
        v1[i] = v2[i] = ss.vi[i];
        v3[i] = ss.vi[i] + ss.vi[i];
    }
}

cl::cl(st1 s1, long ar2[])
{
    for (int i = 0; i < 4; i++)
    {
        v1[i] = v2[i] = s1.vi[i];
        v3[i] = ar2[i];
    }
}

cl cl::elab1(char ar1[], st2 s2)
{
    st1 s1;

    for (int i = 0; i < 4; i++)
    {
        s1.vi[i] = ar1[i];
    }

    cl cla(s1);

    for (int i = 0; i < 4; i++)
    {
        cla.v3[i] = s2.vd[i];
    }

    return cla;
}
```

```
#*******************************************************************************
# File: es1.s
#       Contains the Assembly translation for es1.cpp.
#
# Author: Rambod Rahmani <rambodrahmani@autistici.org>
#         Created on 10/09/2019.
#*******************************************************************************

#-------------------------------------------------------------------------------
.TEXT
.GLOBAL _ZN2clC1E3st1                                            # cl:cl(st1 ss)
#-------------------------------------------------------------------------------
# activation record:
# ------------------
#  i           -16
#  ss          -12
#  this        -8
#  %rbp        0
#-------------------------------------------------------------------------------
_ZN2clC1E3st1:
# set stack locations labels
    .set this, -8
    .set ss,   -12
    .set i,    -16

# prologue: activation frame
    pushq %rbp
    movq  %rsp, %rbp
    subq  $16, %rsp                   # reserve stack space for actual arguments

# copy actual arguments to the stack
    movq %rdi, this(%rbp)
    movl %esi, ss(%rbp)

# for loop initialization
    movl $0, i(%rbp)                  # i = 0
    movq $0, %rax                     # clear out %rax

for:
    cmpl $4, i(%rbp)                  # check if i < 4
    jge  finefor                      # end for loop (i >= 4)

# for loop body
    movq   this(%rbp), %rsi
    movslq i(%rbp), %rcx
    leaq   ss(%rbp), %rdx               # &ss -> %rdx
    movb   (%rdx, %rcx, 1), %al         # ss.vi[i] -> %al
    movb   %al, (%rdi, %rcx, 1)         # v1[i] = ss.vi[i]
    movb   %al, 4(%rdi, %rcx, 1)        # v2[i] = ss.vi[i]
    addl   %eax, %eax                   # ss.vi[i] + ss.vi[i] -> %eax
    movq   %rax, 8(%rdi, %rcx, 8)       # v3[i] = ss.vi[i] + ss.vi[i]

    incl i(%rbp)                      # i++
    jmp  for                          # loop again

finefor:
    movq this(%rbp), %rax             # return intialized object address
    leave                             # movq %rbp, %rsp; popq %rbp
    ret

#-------------------------------------------------------------------------------
.GLOBAL _ZN2clC1E3st1Pl                             # cl::cl(st1 s1, long ar2[])
#-------------------------------------------------------------------------------
# activation record:
# ------------------
#  i            -28
#  &ar2         -24
#  s1           -12
#  this         -8
#  %rbp         0
```

```
#-------------------------------------------------------------------------------
_ZN2clC1E3st1Pl:
# set stack locations labels
    .set this, -8
    .set s1,   -12
    .set ar2,  -24
    .set i,    -28

# prologue: activation frame
    pushq %rbp
    movq  %rsp, %rbp
    subq  $28, %rsp                   # reserve stack space for actual arguments

# copy actual arguments to the stack
    movq %rdi, this(%rbp)
    movl %esi, s1(%rbp)
    movq %rdx, ar2(%rbp)

# for loop initialization
    movl $0, i(%rbp)                  # i = 0

for1:
    cmpl $4, i(%rbp)                  # check if i < 4
    jge  finefor1                     # end for loop (i >= 4)

# for loop body
    movq   this(%rbp), %rdi
    movslq i(%rbp), %rcx
    leaq   s1(%rbp), %rdx             # &s1 -> %rdx
    movb   (%rdx, %rcx, 1), %al       # s1.vi[i] -> %al
    movb   %al, (%rdi, %rcx, 1)       # v1[i] = s1.vi[i]
    movb   %al, 4(%rdi, %rcx, 1)      # v2[i] = s1.vi[i]
    movq   ar2(%rbp), %rsi            # &ar2 -> %rsi
    movq   (%rsi, %rcx, 8), %rbx      # ar2[i] -> %rbx
    movq   %rbx, 8(%rdi, %rcx, 8)     # v3[i] = ar2[i]

    incl i(%rbp)                      # i++
    jmp  for1                         # loop again

finefor1:

    movq this(%rbp), %rax             # return initialized object address
    leave                             # movq %rbp, %rsp; popq %rbp
    ret

#-------------------------------------------------------------------------------
.GLOBAL _ZN2cl5elab1EPc3st2                   # cl cl::elab1(char ar1[], st2 s2)
#-------------------------------------------------------------------------------
# activation record:
# ------------------
#  i            -92
#  cla.v1/v2    -88
#  cla.v3[0]    -80
#  cla.v3[1]    -72
#  cla.v3[2]    -64
#  cla.v3[3]    -56
#  s1           -48
#  s2 [MSB]     -40
#  s2 [LSB]     -32
#  &ar1         -24
#  this         -16     <- this (cl object) address
#  indo         -8      <- leave returned cl object address here
#  %rbp         0
#-------------------------------------------------------------------------------
_ZN2cl5elab1EPc3st2:
# set stack locations labels
    .set indo,   -8
    .set this,   -16
    .set ar1,    -24
    .set s2,     -40
    .set s1,     -48
    .set cla_v3, -80
    .set cla_v2, -84
    .set cla_v1, -88
    .set i,      -92

# prologue: activation frame
    pushq %rbp
    movq  %rsp, %rbp
    subq  $96, %rsp                    # reserve space stack for actual arguments

# copy actual arguments to the stack
    movq %rdi, indo(%rbp)
    movq %rsi, this(%rbp)
    movq %rdx, ar1(%rbp)
    movq %rcx, s2(%rbp)
    movq %r8, -32(%rbp)

# for loop 1 initialization
    movl $0, i(%rbp)                   # i = 0

for2:
    cmpl $4, i(%rbp)                   # check if i < 4
    jge  finefor2                      # end for loop (i >= 4)

# for loop 1 body
    movslq i(%rbp), %rcx               # i --64ext--> %rcx
    movq   ar1(%rbp), %rdi             # &ar1 -> %rsi
    movb   (%rdi, %rcx, 1), %al        # ar1[i] -> %al
    leaq   s1(%rbp), %rsi              # &s1 -> %rax
    movb   %al, (%rsi, %rcx, 1)        # s1.vi[i] = ar1[i]

    incl   i(%rbp)                     # i++
    jmp    for2                        # loop again

finefor2:

# prepare actual arguments to call constructor
    leaq  cla_v1(%rbp), %rdi           # leave &this in %rdi
    movl  s1(%rbp), %esi               # leave ss in %rsi
    call  _ZN2clC1E3st1                # cl cla(s1);

# for loop 2 initialization
    movl  $0, i(%rbp)

for3:
    cmpl $4, i(%rbp)
    jge  finefor3

# for loop 2 body
    movslq i(%rbp), %rcx               # i -> %rcx
    leaq   s2(%rbp), %rdx              # &s2 -> %rax
    movl   (%rdx, %rcx, 4), %ebx       # s2.vd[i] -> %ebx
    movslq %ebx, %rbx
    movq   %rbx, cla_v3(%rbp, %rcx, 8)

    incl  i(%rbp)                      # i++
    jmp   for3                         # loop again

finefor3:

# copy return object from stack to the address in indo
    leaq cla_v1(%rbp), %rsi            # rep movsq source address
    movq indo(%rbp), %rdi              # rep movsq destination address
    movabsq $5, %rcx                   # rep movsq repetitions
    rep movsq                          # rep movsq, [0]
    movq indo(%rbp), %rax              # return initialized object address

    leave                              # movq %rbp, %rsp; popq %rbp;
    ret
```

```
#*******************************************************************************

################################################################################
# [0]
# Copies the quad word address by %rsi into the location addressed by %rdi. It
# will then increment both %rsi and %rdi and repeat. The number of repetitions
# is set using %rcx.
################################################################################
```

```
/**
 * File: prova1.cpp
 *       This file contains a developer harness test for es1.s.
 *
 *       Compile with:
 *          g++ -o es1 -fno-elide-constructors es1.s prova1.cpp
 *
 *       Test your result with:
 *          ./es1 | diff - es1.out
 *
 * Author: Rambod Rahmani <rambodrahmani@autistici.org>
 *         Created on 10/09/2019.
 */

#include "cc.h"

/**
 * Developer harness test.
 *
 * @param  argc  command line arguments counter.
 * @param  argv  command line arguments.
 *
 * @return       execution exit code.
 */
int main(int argc, char * argv[])
{
    st1 s1 = { 1,2,3,4 };

    st2 s2 = { 5,6,7,8 };

    char a1[4] = { 11,12,13,14 };

    long a2[4] = {15,16,17,18 };

    cl cla1(s1);

    cla1.stampa();

    cl cla2(s1, a2);

    cla2.stampa();

    cla1 = cla2.elab1(a1, s2);

    cla1.stampa();

    //system("pause");
}
```

```
1 2 3 4
1 2 3 4
2 4 6 8

1 2 3 4
1 2 3 4
15 16 17 18

11 12 13 14
11 12 13 14
5 6 7 8
```

```
// EXTENSION 2016-07-06

/**
 * Interrupt type declaration for primitive ceread().
 * This interrupt type is the one called by the primitive implementation in
 * utente/utente.s and must be handled in io/io.s.
 */
#define IO_TIPO_CEREAD          0x79

#define IO_TIPO_CEWRITE         0x7a

// EXTENSION 2016-07-06
// EXTENSION 2016-07-06

/**
 * Declare primitive here in order for it to be accessible from the user module.
 *
 * This primitive reads from the RBR register of the CE device having the given
 * id into the provided buffer until the char 'stop' is retrieved or the maximum
 * number of bytes is reached.
 *
 * @param  id      CE device id;
 * @param  buf     destination buffer address;
 * @param  quanti  number of bytes to retrieve/number of bytes actually
 *                 transferred;
 * @param  stop    stop char.
 */
extern "C" void ceread(natl id, char *buf, natl& quanti, char stop);

// EXTENSION 2016-07-06
```

```
# EXTENSION 2016-07-06

#-------------------------------------------------------------------------------
# Definition of the ceread primitive.
# Just calls the IO_TIPO_CEREAD interrupt and returns.
# The IO_TIPO_CEREAD interrupt is handled in io/io.s/a_ceread ->
# io/io.cpp/c_ceread()
#-------------------------------------------------------------------------------
.GLOBAL ceread
#-------------------------------------------------------------------------------
ceread:
        int $IO_TIPO_CEREAD
        ret

# EXTENSION 2016-07-06
# SOLUTION 2016-07-06

    fill_io_gate    IO_TIPO_CEREAD  a_ceread

# SOLUTION 2016-07-06

# SOLUTION 2016-07-06

#------------------------------------------------------------------------------
.EXTERN c_ceread                                # C++ implementation for a_ceread
#------------------------------------------------------------------------------
# IDT subroutine for the IO_TIPO_CEREAD interrupt.
a_ceread:
    cavallo_di_troia %rsi           # check buffer address
    cavallo_di_troia %rdx           # check quanti address
    cavallo_di_troia2 %rdx $4       # quanti will contain a integer (4 bytes)
    movl (%rdx), %r9d                           # move buffer length and zero out r9
    cavallo_di_troia2 %rsi %r9      # check the buffer entire length
    call c_ceread                   # call C++ int. primitive implementation
    iretq                           # return from interrupt

# SOLUTION 2016-07-06
```

```
// SOLUTION 2016-07-06

/**
 * CE device descriptor. CE devices are PCI devices and can not work in bus
 * mastering (DMA). Transfers must be handled reading from the RBR register each
 * byte.
 */
struct des_ce
{
    // control register address
    ioaddr iCTL;

    // status register address
    ioaddr iSTS;

    // RBR register address
    ioaddr iRBR;

    // synchronization semaphore
    natl sync;

    // mutex semaphor
    natl mutex;

    // destination buffer virtual address
    char * buf;

    // number of bytes to be transferred
    natl quanti;

    // char used to stop the transfer
    char stop;
};
// SOLUTION 2016-07-06

// EXTENSION 2016-07-06
/**
 * Maximum number of CE devices to be initialized at boot.
 */
static const int MAX_CE = 16;

/**
 * CE devices decriptors array.
 */
des_ce array_ce[MAX_CE];

/**
 * Next CE device id to be initialized.
 */
natl next_ce;

// EXTENSION 2016-07-06

// SOLUTION 2016-07-06
/**
 * Called by the IO_TIPO_CEREAD interrupt handler a_ceread in io/io.s.
 *
 * Retrieves from the RBR register of the given CE device a number of bytes
 * equal to ’quanti’ into the destination buffer. If the stop char is retrieved
 * the transfer will be stopped before reaching the bytes limit.
 *
 * @param  id      CE device id;
 * @param  buf     destination buffer address;
 * @param  quanti  number of bytes to retrieve;
 * @param  stop    stop char.
 */
extern "C" void c_ceread(natl id, char * buf, natl& quanti, char stop)
{
    // check if the given CE device id is valid
    if (id >= next_ce)
    {
        // if not, print a warning log message
        flog(LOG_WARN, "CE Device %d does not exit.");

        // abort current process under execution
        abort_p();
    }

    // retrieve CE device descriptor
    des_ce *c = &array_ce[id];

    // wait for the CE device mutex
    sem_wait(c->mutex);

    // set destination buffer address
    c->buf = buf;

    // set number of bytes to be transferred
    c->quanti = quanti;

    // set stop char
    c->stop = stop;

    // write to the control register: enable interrupt requests
    outputb(1, c->iCTL);

    // wait for the synchronization sempahore: set in estern_ce
    sem_wait(c->sync);

    // set number of bytes actually transferred
    quanti -= c->quanti;

    // signal mutex semaphore
    sem_signal(c->mutex);
}

/**
 * Called everytime the CE device having the given id sends an interrupt
 * request.
 *
 * @param  id  the id of the CE device sending the interrupt request.
 */
extern "C" void estern_ce(int id)
{
    // retrieve CE device descriptor
    des_ce *c = &array_ce[id];

    // RBR register temp destination buffer
    natb b;

    // this infinite for loop is needed because once the wfi() is done sending
    // the EOI to the APIC it will also schedule a new process; when a new
    // interrupt request is received from this ce device this process will wake
    // up again and start from where it was ended: without the for loop the
    // function will just end resulting in a dead lock
    for (;;)
    {
        // stop CE device interrupt requests
        outputb(0, c->iCTL);

        // read RBR register content: interrupt request ak
        inputb(c->iRBR, b);

        // write transferred byte
        *c->buf++ = b;

        // decrease number of bytes to be transferred
        c->quanti--;

        // check if either the number of bytes to be transferred has been
        // reached or the stop char has been retrieved
        if (c->quanti == 0 || b == c->stop)
        {
            // if so, signal synchronization semaphore
            sem_signal(c->sync);
                }
        else
        {
            // otherwise, enable interrupt requests
            outputb(1, c->iCTL);
        }

        // send End Of Interrupt to APIC
        wfi();
    }
}
//   SOLUZIONE 2016-07-06 )

// EXTENSION 2016-07-06
/**
 * Initializes the CE devices on the PCI bus. Called at the end of the I/O
 * module initialization.
 */
bool ce_init()
{
    // loop through the PCI bus devices
    for (natb bus = 0, dev = 0, fun = 0;
         pci_find_dev(bus, dev, fun, 0xedce, 0x1234);
         pci_next(bus, dev, fun))
    {
        // check the number of retrieved CE devices
        if (next_ce >= MAX_CE)
        {
            // print warning log message
                flog(LOG_WARN, "Too many CE devices.");

            // exit for loop
            break;
        }

        // retrieve pointer to available CE device descriptor
        des_ce *ce = &array_ce[next_ce];

        // retrieve base register content
        natw base = pci_read_confl(bus, dev, fun, 0x10);

        // set bit n.0 to 0: retrieve base register address
        base &= ~0x1;

        // set control register address: base
        ce->iCTL = base;

        // set status register address: base + 4
        ce->iSTS = base + 4;

        // set RBR register address: base + 8
        ce->iRBR = base + 8;

        // initialize synchronization semaphore
        ce->sync = sem_ini(0);

        // initialize mutex sempahore
        ce->mutex = sem_ini(1);

        // retrieve PCI device APIC pin
        natb irq = pci_read_confb(bus, dev, fun, 0x3c);

        // activate external process
        activate_pe(estern_ce, next_ce, PRIO, LIV, irq);
        // log CE device info
        flog(LOG_INFO, "ce%d %2x:%1x:%1x base=%4x IRQ=%d", next_ce, bus, dev, fun, base,
irq);

        // increase CE devices counter
        next_ce++;
    }

    // return true: initialization successful
    return true;
}
// EXTENSION 2016-07-06

////////////////////////////////////////////////////////////////////////////////
//                 INIZIALIZZAZIONE DEL SOTTOSISTEMA DI I/O                   //
////////////////////////////////////////////////////////////////////////////////

// inizializza i gate usati per le chiamate di IO
//
extern "C" void fill_io_gates(void);

extern "C" natl end;
// eseguita in fase di inizializzazione
//
extern "C" void cmain(int sem_io)
{

        fill_io_gates();
        mem_mutex = sem_ini(1);
        if (mem_mutex == 0xFFFFFFFF) {
                flog(LOG_ERR, "impossible creare semaforo mem_mutex");
                abort_p();
        }
        unsigned long long end_ = (unsigned long long)&end;
        end_ = (end_ + DIM_PAGINA - 1) & ~(DIM_PAGINA - 1);
        heap_init((void *)end_, DIM_IO_HEAP);
        if (!console_init())
                abort_p();
        if (!com_init())
                abort_p();
        if (!hd_init())
                abort_p();

// EXTENSION 2016-07-06

    // initialize CE devices
    if (!ce_init())
    {
        // abort the current process if the initialization does not succeed
        abort_p();
    }

// EXTENSION 2016-07-06

        sem_signal(sem_io);
        terminate_p();
}
```