#include "libepc.h"
#include "dos.h"

extern unsigned inportdw(int port_addr);
extern unsigned outportdw(int port_addr, int data_out);

int main(void)
{
    int i, j, k, m, r, x, entry_r;
    char scan_code, ascii_code;
    unsigned data_port, old_port, data_port_ba;
    unsigned data_port_barq;
    union pci_dev_addr{
        unsigned int pci_info;
        struct config_fld{
            unsigned type:2;
            unsigned offset:6;
            unsigned funct:3;
            unsigned dev:5;
            unsigned bus:8;
            unsigned res:7;
            unsigned enabl:1;
            }pci_fields;
    }addr_port;
}
// clear the screen, and prepare for output on line 1

ClearScreen(0x07) ;
r = 1;

// old_port used to compare common function info

old_port = 0;

// look for just 10 buses (PCI could have up to 256)

for(k = 0; k<10; k++) { // bus index
  for(i = 0; i<32; i++) { // device index
    for(j = 0; j<8; j++) { // function index
      addr_port.pci_fields.type = 0;
      addr_port.pci_fields.offset = 0;
      addr_port.pci_fields.funct = j;
      addr_port.pci_fields.dev = i;
      addr_port.pci_fields.bus = k;
      addr_port.pci_fields.res = 0;
      addr_port.pci_fields.enabl = 1;
    }
  }
}
// set up address port at 0xCF8 offset 0 (dev|vend)
outportdw(0x0CF8, addr_port.pci_info);

// now grab data port with device|vendor 4 byte dword
data_port = inportdw(0x0CFC);

// if data port is full of 1s, no device is present
if(data_port == 0xFFFFFFFF)continue;

// if data port is the same a previous function value ignore
if(data_port == old_port)continue;

// we have valid device|vendor dword, prepare to print
SetCursorPosition(r++, 5);
// use top half of screen for d|v, go back to line 1 if past 10

if(r > 11)r=1;

// print device|vendor dword in hex

PutString("Bus: ");
PutUnsigned(k, 16, 2);
PutString(" Device: ");
PutUnsigned(i, 16, 2);
PutString("  Funct: ");
PutUnsigned(j, 16, 1);
PutString("  Dev|Vendor: ");
PutUnsigned(data_port, 16, 8);

// remember current data port value for comparison with same
// device, different function in next port data

old_port = data_port;
// does user want to see base registers for this object?
PutString("  BA Reg ? [y|n]: ");

// get ready to read input scan codes
enable();
while(! ScanCodeRdy()){
    x++;
}
scan_code = GetScanCode();

// convert down-key scan code to ascii code
ascii_code = ScanCode2Ascii(scan_code);

// echo user reply to [y|n]
PutChar(ascii_code);
// now get up-key scan code to clear the input buffer

while(! ScanCodeRdy()){    
    x++;                  
}

scan_code = GetScanCode();

// check if down key scan code was NOT a y (yes)
// if user did not answer yes, go to the next PCI object

if(ascii_code != 'y')continue;

// if user did answer yes, prepare to print out base address
// register information for this PCI object

else{
    entry_r = r;
}
// remember print line number in upper half of screen and prepare
// to print out base address register info in lower part of screen
// clear the 13 lower screen lines to all spaces
    r = 12;
    for(m = 0; m<13; m++){
        SetCursorPosition(r++, 0);
        PutString("          ……… ");
    }

// go to line 13, print d|v value and then the base reg info

    r = 12;
    SetCursorPosition(r++, 0);
    PutString(" BASE ADDRESS REGISTERS FOR: ");
    PutUnsigned(data_port, 16, 8);

} //else
for(m=0; m<6; m++){
    addr_port.pci_fields.offset = m+4;
    outportdw(0x0CF8, addr_port.pci_info);
    data_port_ba = inportdw(0x0CFC);
    SetCursorPosition(r++, 7);
    PutString("Base Addr Register is: ");
    PutUnsigned(m, 16, 2);
    PutString(" Base Addr is: ");
    PutUnsigned(data_port_ba, 16, 2);
... ETC. ...
SECTION .text
BITS32

GLOBAL _inportdw
_inportdw:   MOV   DX,[ESP+4]  ; port
             IN    EAX,DX
             RET

GLOBAL _outportdw
_outportdw:  MOV   DX,[ESP+4]  ; port
              MOV   EAX,[ESP+8] ; data
              OUT   DX,EAX
              RET
Building Bootable Floppy

• Unzip the file djdev203.zip in the djgpp\binary directory (assuming c:\djgpp)
• In a dos box:
  – set djgpp=c:\djgpp\binary\djgpp.env
  – set PATH=c:\djgpp\binary\bin;%PATH%
• Unzip the file nasm098p.zip in the nasm\binary directory
  – This provides nasm.exe
Building Bootable Floppy (cont’d)

• In the directory `c:\djgpp\binary\bin` find and run the ldscript utility:
  – ldscript > link.cmd
  – link.cmd must now be used in linking your applications

• Compile your c code using the gcc utility:
  – gcc –c myfile.c
  – This will yield myfile.o

• Assemble your assembly code with nasm:
  – nasm –f coff myasm.asm
  – This will yield myasm.o
Building Bootable Floppy (cont’d)

- In the directory with your compiled and assembled .o files
  - Use the ld command to link a binary:
  - This creates a file called embedded.bin in the directory
  - ld myfile.o myasm.o –Tlink.cmd –u start –Map link.map
- Find the directory c:\djgpp\bootload, put a floppy in the drive, and in a dos box run the command:
  - copyboot a:
- From the build directory run the dos command:
  - copy embedded.bin a:
- Boot the floppy and your code will run