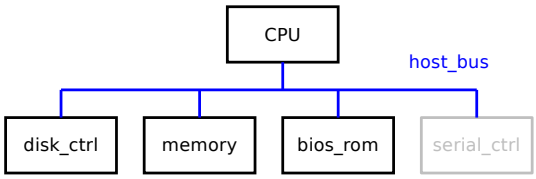
Übungen zu Virtuelle Maschinen, Aufgabe 1

Volkmar Sieh

WS2015/2016

1 Übersicht

In Aufgabe 1 soll eine virtuelle Maschine entwickelt werden, welche eine stark begrenzte Untermenge der Befehle der x86-Architektur ausführen kann. Neben dem CPU-Simulator sollen zusätzlich Simulatoren für einen Festplattencontroller, sowie einen RAM und einen ROM-Baustein implementiert werden. Diese virtuelle Hardware soll der Einfachheit halber direkt mit dem Host-Bus verbunden werden.



2 Organisatorisches

Bitte senden Sie Ihre Lösung bis zum 30.11.2015 per Mail an i4vm@cs.fau.de. Die Projektdateien finden Sie unter
https://www4.cs.fau.de/Lehre/WS15/V_VM/Uebungen/aufgabe1.tar.gz.

3 Hardware-Simulatoren

Implementieren Sie Hardware-Simulatoren, die folgenden Modellbeschreibungen genügen. Der Einfachheit halber, soll sämtliche virtuelle Hardware direkt mit dem Host-Bus verbunden werden. Die CPU agiert hierbei als Bus-Master, die einzelnen Komponenten als Slave-Geräte.

3.1 Festplattencontroller und Festplatte

Der Festplattencontroller soll ab Adresse `0xD000` angesteuert werden können. Von den 32 Adressleitungen des Host-Busses, werden die oberen 22 Leitungen als Chip-Select Leitungen genutzt.

Wird eine Adresse von `0` bis `3` auf den verbleibenden 10 Adressleitungen selektiert, so kann hiermit lesend und schreibend auf das Blocknummerregister zugegriffen werden. Der Inhalt diese 32-Bit Registers soll in *big-endian*-Form dargestellt werden. Das Blocknummerregister wird bei Lese- und Schreiboperationen ausgewertet.

Bei Adresse `7` befindet sich ein 8-Bit Fehlerregister, welches gelesen oder beschrieben werden kann.

Der Adressbereich `0x200`–`0x3FF` ist ein 512 Byte großer Puffer. Dieser Puffer kann sowohl gelesen als auch beschrieben werden.

Selektieren der Adresse `0xb` setzt die Lese-/Schreiblogik des Controllers in Gang. Wird eine Eins an diese Adresse geschrieben, so wird der Blockpuffer auf die Platte geschrieben, und zwar an den Block, auf den das Blocknummerregister zeigt. Ist dieser Block ungültig, so wird das Fehlerregister mit einer Eins beschrieben.

Wird eine Null in an Adresse `0xb` geschrieben, so soll der ausgewählte Block von der Platte in den Blockpuffer geladen werden. Auch hier muss zunächst die Gültigkeit des Blocks geprüft werden, und gegebenenfalls das Fehlerregister mit einer Eins aktualisiert werden.

Der Einfachheit halber, soll die Implementierung auf jegliche Zeitverzögerungen verzichten. Das heißt, dass der Blockpuffer unmittelbar nach Ansteuern der Leselogik mit den neuen Daten befüllt ist. Ein Schreiben soll ebenfalls sofort erfolgen.

Die Festplatte soll eine Größe von `5 MiB` besitzen. Der letzte Aufrufparameter von `disk_ctrl_create` benennt eine Datei, welche den Inhalt der Festplatte enthält.

3.2 Speicher

Die Virtuelle Maschine soll über zwei Speicherbausteine verfügen. Zum einen kann ein 4 KiB ROM ein rudimentäres BIOS aufnehmen, welches einen Bootloader von Platte laden kann, zum anderen soll die virtuelle Maschine über 32 KiB RAM für Benutzerprogramme und Daten verfügen.

3.2.1 RAM

Der 32 KiB RAM-Baustein soll ab Adresse `0x0000` angesteuert werden können. Tip: Initialisieren Sie den Inhalt des RAM-Bausteins sinnvoll.

3.2.2 Bios ROM

Der ROM-Baustein soll ab Adresse `0xE000` verfügbar sein und eine Größe von `4 KiB` besitzen. Der ursprüngliche Inhalt des ROMs soll aus einer Datei geladen werden. Der Name der BIOS-Datei soll hierbei als letzter Parameter der `bios_rom_create` Funktion angegeben werden können. Ist diese Datei kleiner als das ROM, so soll der Rest des ROMs geeignet initialisiert werden. Schreibzugriffe auf das BIOS-ROM sollen ignoriert werden.

3.3 CPU Simulator

Der CPU-Simulator soll – zumindest in kleinen Teilen – die *x86*-ISA simulieren können. Zur Vereinfachung genügt es, lediglich den *Protected Mode* zu unterstützen. Ebenso sollen zunächst keinerlei Exceptions oder Interrupts simuliert werden.

Da der Befehlssatz der *x86*-Architektur sehr umfangreich ist, genügt es, die folgenden Befehle zu simulieren:

- `mov`
- `cmp`
- `jb`
- `je`
- `jne`
- `call`
- `ret`
- `xor`
- `add`
- `inc`
- `dec`
- `jmp`
- `ljmp`
- `hlt`

Nach Initialisieren des CPU-Simulators, soll der Instruction Pointer `eip` auf Adresse `0xE000`, also die erste Adresse des ROMs zeigen. Mit der Funktion `cpu_step` soll der CPU-Simulator die jeweils nächste Instruktion ausführen. Der Rückgabewert zeigt an,

ob eine weitere Instruktion ausgeführt werden kann (`true`), oder ob sich die CPU in einem Halt-Zustand befindet (`false`).

In den Projektdateien finden sich einige Assembler-Programme, mit denen Sie Ihren CPU-Simulator testen können.

4 Projektdateien

Im Folgenden finden Sie eine Kurzübersicht über die Projektdateien. Sämtliche Dateien dürfen Sie nach Belieben verändern.

- `bus/sig_host_bus.h,c`: Implementierung des Host-Bus-Protokolls. Der Host-Bus verfügt über 32 Adressleitungen aber lediglich über 8 Datenleitungen. Das bedeutet, dass Daten lediglich Byte-weise übertragen werden können.
- `comp/serial_ctrl.h,c`: Eine rudimentäre serielle Schnittstelle, die ab Adresse `0xD800` angesteuert werden kann. In der vorliegenden Implementierung können lediglich Bytes gesendet werden, Empfang ist derzeit nicht vorgesehen. Das Senderegister befindet sich an Adresse `0xD800`. Sämtliche Bytes, die in das Senderegister geschrieben werden, werden auf `stdout` ausgegeben. Somit können Programme im Simulator die serielle Schnittstelle zu Debug-Zwecken nutzen.
- `setup.c`: Diese Datei stellt ein kleines Testprogramm zur Verfügung, welches die einzelnen Komponenten der virtuellen Maschine erzeugt und den CPU-Simulator in einer Schleife ausführt.
- `src/comp/roms/`: Assemblerprogramme, die zum Testen des CPU-Simulators verwendet werden können. Diese Programme sind dazu geeignet, im BIOS-ROM ausgeführt zu werden:
  - `src/comp/roms/bios_print_simple.S`: Einfaches Testprogramm, gibt „Hallo Welt“ auf der seriellen Schnittstelle aus.
  - `src/comp/roms/bios_print.S`: Noch ein „Hallo Welt“ Programm, diesmal jedoch in Form einer Schleife anstelle von sequentiellen Anweisungen.
  - `src/comp/roms/bios_boot.S`: Minimales BIOS-Boot-Programm, welches den ersten Sektor der Festplatte in den Speicher lädt und anschließend den geladenen Code anspringt.
- `src/comp/test/`: Hier befinden sich weitere Assemblerprogramme zum Testen des CPU-Simulators. Diese Programme sind dazu geeignet aus dem Boot-Sektor der Festplatte geladen zu werden. Zusätzlich können diese Programme auch eigenständig auf Linux-Systemen ausgeführt werden, sofern sie mit der Bibliothek `linux_libs.S` gebunden werden.
  - `hallo.S`: Eine weitere Variante des „Hallo Welt“ Programms.
  - `vm_libs.S`: Ausgabebibliothek für die virtuelle Maschine (mittels serieller Schnittstelle).
  - `linux_libs.S`: Ausgabebibliothek, welche Linux System-Calls verwendet.
  - `arith.S`: Einfache Rechenoperationen für BCD-artige lange Zahlen. Berechnet die Fakultät für 250 bei einer Zahlenlänge von 500 Stellen.

    Achtung: Dieses Programm passt nicht mehr in einen Block. Gegebenenfalls müssen Sie den Bootloader anpassen.