1 Übersicht

In Aufgabe 3 soll die bisher entwickelte virtuelle Maschine um eine *MMU* erweitert werden. Hierbei soll zunächst Segmentierung realisiert werden.

2 Organisatorisches

Bitte senden Sie Ihre Lösung bis zum 21.01.2015 per Mail an i4vm@cs.fau.de. Die Projektdateien mit Test-Programmen finden Sie unter
https://www4.cs.fau.de/Lehre/WS14/V_VM/Uebungen/aufgabe3.tar.gz.

3 CPU-Simulator

3.1 MMU - Segmentierung

Erweitern Sie den CPU-Emulator um die Möglichkeit, Segmentierung nutzen zu können. Wie auch in den vorherigen Aufgaben, soll der CPU-Simulator lediglich den *Protected Mode* unterstützen. Zu Beginn sollen alle Segmentregister mit der Basisadresse `0x0000` und dem Limit `0xFFFF` initialisiert werden. Des Weiteren sollen alle Segmentregister Lese- und Schreibberechtigungen enthalten, das Code-Segmentregister zusätzlich die `Execute`-Berechtigung.

Somit zeigt die Startadresse, welche der *Instruction Pointer* zu Beginn enthalten soll, auf den Anfang des BIOS Roms.

Zur Vereinfachung genügt es, lediglich bei folgenden Verletzungen der Segmentierung Exceptions zu generieren:

- Überschreitung des Limits
- Laden eines Datensegmentdeskriptors in das `CS`-Register
- Laden eines Segmentdeskriptors, welcher auf ein nicht vorhandenes Segment zeigt.

Andere Verletzungen, wie z.B. Schreiben in ein nicht schreibbares Segment oder Lesen eines nicht lesbaren Segments müssen nicht geprüft werden. Zur weiteren Vereinfachung brauchen lediglich Segmente realisiert werden, welche

- sich in Ring 0 befinden
- als 32-Bit markiert sind
- Code- oder Datensegmente sind
- nicht als „Expand-Down“ markiert sind
- nicht als „Conforming“ markiert sind

3.2 Weitere Opcodes

Die in dieser Aufgabe zu realisierende Segmentierung bietet die Möglichkeit, Betriebssystemcode vor Benutzerprogrammen zu schützen, zumindest sofern die Benutzerprogramme nicht böswilligerweise die Segmentregister umsetzen. Um einen sauberen Einstiegspunkt für Systemaufrufe zu schaffen, ist der `int`-Befehl zu implementieren. Außerdem müssen die Befehle `push` und `pop` realisiert werden, um sicherzustellen, dass Interrupts den Prozessorzustand aus Sicht der Benutzerprogramme nicht stören. Schließlich wird der Befehl `and` benötigt.

4 Testprogramm

Das Testprogramm finden Sie im Verzeichnis `test-segmentation`. Benötigte Bibliotheken zur Ein-/Ausgabe liegen in den Verzeichnissen `lib_real` bzw. `lib_vm`. `lib_real` enthält hierbei Varianten, die mit real existierender Hardware bzw. mit dem Linux-Kern kommunizieren können, `lib_vm` die entsprechenden Implementierungen für die vereinfachte VM.

Verzeichnis `test-segmentation`:

- `bios_boot.S`: Implementierung eines erweiterten Boot-Loaders, welcher die Segmentregister initialisiert und die ersten 56 Sektoren der Festplatte in den Hauptspeicher ab Adresse `0x0000` kopiert. Nach dem Kopieren wird an Adresse `0x0000` gesprungen.
- `main.S`: „Betriebssystem“, welches die Hardwarekomponenten initialisiert und ein Menü anzeigt, mit dem jeweils ein Prozess gestartet werden kann. Jeder auswählbare Prozess wird hierbei in eigenen Code-/Datensegmenten ausgeführt.
- `hallo.S`: „Hallo Welt“ als Prozess.
- `arith.S`: Berechnen der Fakultät als Prozess.
- `gp.S`: Kleiner Prozess, welcher auf eine Adresse außerhalb des Segmentlimits zugreift.
- `Makefile`: Baut sowohl die ein BIOS Rom mit dem Bootloader also auch ein Festplatten-Image. Die Prozesse liegen hierbei an den relativen Offsets `0x2000` (hallo.S), `0x3000` (arith.S) und `0x4000` (gp.S).
- Unterverzeichnis `x86_cdrom`: Das dortige `Makefile` erzeugt ein CD-ISO-Image, welches die Testprogramme mit den Bibliotheken für echte Hardware als Bootloader enthält. Somit sollte es auf einem realen System funktionieren.