Bitte bearbeiten Sie die folgenden Aufgaben in Zweier- oder Dreiergruppen. Wenn Sie Feedback zu Ihren Lösungen haben möchten, notieren Sie Ihre Lösungen in einem Textdokument und schicken Sie es mir per E-Mail (bitte keine Mehrfachabgabe identischer Lösungen).

6. Universalrechner, von Neumann vs. Harvard

a) Beschreiben Sie in eigenen Worten, wie sich die von-Neumann-Architektur von der Harvard-Architektur unterscheidet. Nennen Sie für beide Architekturen Vor- und Nachteile.

b) Im Allgemeinen betrachtet man den Prozessor auf einer Grafikkarte nicht als „Universalrechner". GPGPU (General Purpose Computing on Graphics Processing Units) ist ein Ansatz, Grafikkarten für komplexe Berechnungen zu verwenden, die nichts mit Grafikdarstellung zu tun haben; Beispiele dafür sind die Architekturen OpenCL und CUDA (Nvidia). Ist damit die prinzipielle Trennung zwischen „normalen" CPUs und GPUs noch sinnvoll? Wenn Sie für die Beantwortung mehr Informationen benötigen, informieren Sie sich zu Hause über GPGPU (z. B. unter http://gpgpu.org/ oder bei Wikipedia).

7. Spezialregister

Die in den meisten CPUs vorhandenen Register *Befehlszähler* (PC, Program Counter) und *Stack Pointer* (SP) können Sie nicht wie „normale" Register direkt verändern. Trotzdem gibt es Befehle, die diese Register beeinflussen.

a) Wie können Sie einen bestimmten Wert in das PC-Register schreiben?

b) Warum ist es meist nicht sinnvoll, analog den Stack Pointer von Hand zu setzen?

8. Stack-Maschine

Bei einer Stack-Maschine können Sie mit `PUSH Adresse` den Inhalt der Speicherzelle Adresse auf den Prozessor-Stack schreiben und mit `POP Adresse` den obersten Wert im Prozessor-Stack entfernen und in der Speicherzelle speichern. Die vier Grundrechenarten rufen Sie über die Befehle `ADD`, `SUB`, `MUL` und `DIV` auf. Zum Beispiel berechnen Sie das Quadrat der an Adresse 0x1000 gespeicherten Zahl mit den Kommandos

```
PUSH 0x1000
PUSH 0x1000
MUL
POP 0x1001
```

(Der letzte Befehl schreibt das Ergebnis in Speicherzelle 0x1001.)

Schreiben Sie ein Programm, das die Funktion $f(x) = (x^3 - 2x + a) / (x - b)$ berechnet. Sie können dabei davon ausgehen, dass der Wert x (als ein Byte) in der Speicherzelle 0x1000 gespeichert ist; die Konstanten a und b liegen in den Speicherzellen 0x1001 und 0x1002. Weitere Speicherzellen ab 0x1003 können Sie zum Speichern von Zwischenergebnissen verwenden. Rechenfehler (durch Überlauf oder Rundung) können Sie für den Zweck dieser Aufgabe ignorieren; das Programm sollte für x=1, 2, 3 mit a=1, b=0 funktionieren.

9. Register-Memory-Befehle

Die MMIX-CPU beherrscht keine Register-Memory-Befehle, also Operationen, die gleichzeitig auf Register und Speicherzellen zugreifen; stattdessen gibt es nur Load- und Store-Befehle, mit denen Sie Speicherinhalte in eines der 255 Register (oder umgekehrt) kopieren können, und Operationen, die ausschließlich mit Registern arbeiten.

Die (nicht vorhandenen) Befehle

a) `ADD Reg, (Mem)`
(addiere Inhalt der Speicherzelle *Mem* zu Inhalt von Register *Reg* und speichere Wert in Reg)

b) `ADD (Mem1), (Mem2)`
(addiere Inhalt der Speicherzellen *Mem1* und *Mem2* und speichere Wert in Zelle Mem1)

c) `JPEQ (Mem1), (Mem2), Adr`
(wenn in den Speicherzellen *Mem1* und *Mem2* identische Werte stehen, springe zu Adresse *Adr*)

sollen in MMIX nachgebaut werden. Geben Sie geeigneten Code an, der das leistet (wobei Sie davon ausgehen können, dass die Register $200 bis $254 ansonsten nicht verwendet werden). Für Aufgabe **c)** benötigen Sie die MMIX-Instruktion `BZ $Reg, Adr`, die zur Adresse *Adr* springt, wenn das Register $Reg den Wert 0 enthält (BZ: **B**ranch if **Z**ero).

10. Akkumulator-Maschinen

a) Formulieren Sie das Programm aus Aufgabe 3 erneut, diesmal für eine Akkumulator-Maschine. Sie können dabei Befehle der Form

`LOAD Adr`	(lade Inhalt der Speicherzelle *Adr* in Akku)
`STOR Adr`	(speichere Akku-Inhalt in der Speicherzelle *Adr*)
`ADD Adr`	(addiere Inhalt der Speicherzelle *Adr* zu Akku)
`MUL Adr`	(multipliziere Inhalt der Speicherzelle *Adr* mit Akku)

(und entsprechend: `SUB Adr`, `DIV Adr`) verwenden. Es reicht, wenn am Ende das Ergebnis im Akku steht. Die Werte für *x*, *a*, *b* liegen wieder in den Speicherzellen 0x1000, 0x1001 und 0x1002. Höhere Adressen können Sie zum Zwischenspeichern nutzen.

b) Warum verwenden moderne Prozessoren zahlreiche Register und nicht – wie die Akkumulator-Maschinen – ein einziges (den Akku)?

c) Wie sieht das Programm aus Teil a) aus, wenn Sie es für MMIX erstellen? Gehen Sie hier davon aus, dass die Werte von *x*, *a* und *b* bereits in den Registern $1, $2, $3 vorliegen. Register ab $4 stehen Ihnen frei zur Verfügung.

Wenn Sie alle Aufgaben bearbeitet haben, vergleichen Sie Ihre Lösungen mit einer benachbarten Zweiergruppe. Notieren Sie sich ggf. abweichende Lösungsmöglichkeiten.

Eine Musterlösung besprechen wir beim nächsten Vorlesungstermin.

Literatur: MMIX Quick Reference:
http://mmix.cs.hm.edu/doc/mmix-refcard-a4.pdf
(oder Google-Suche nach „MMIX Quick Reference“, 1. Treffer)