Prof. Dr. Christoph Scholl
Dr. Tim Welschehold
Alexander Konrad
Niklas Wetzel

Freiburg, 28.10.2022

Betriebssysteme
Musterlösung zu Übungsblatt 2

Aufgabe 1 (3+1+3+1 Punkte)

In der Vorlesung haben sie eine veränderte Form der aus TI bekannten ReTI kennengelernt, die um essentielle hardwaremäßige Grundlagen zur Implementierung eines Betriebssystems erweitert wurde. Unter anderem enthält die ReTI nun neue Register (z.B. `SP`, welches in die erste freie Speicherzelle des Stack zeigt) sowie zusätzliche Datenpfade.

a) Nehmen Sie an, auf der erweiterten ReTI wird der Befehl

```
STOREIN ACC SP i
```

ausgeführt. Betrachten Sie nur die Execute-Phase (Hinweis: Nach der Fetch-Phase liegt der Operand i im Instruktionsregister I bereit). Welche Treiber müssen hierfür aktiv geschaltet werden? Begründen Sie Ihre Antwort.

b) Diesmal wird auf der erweiterten ReTI der Befehl

```
MOVE IN2 ACC
```

ausgeführt. Betrachten Sie wieder nur die Execute-Phase. Geben Sie die minimale Anzahl an Treibern an, die für den Befehl aktiviert werden müssen und benennen Sie diese Treiber. Begründen Sie Ihre Antwort.

c) Die Befehlsklasse der `COMPUTE`-Befehle wurde in der Vorlesung um sogenannte register-only Befehle erweitert. Ist es möglich mit den vorhandenen Datenpfaden den Befehl

```
ADD ACC IN1
```

zu implementieren? Und falls ja, welche Treiber müssen in der Execute-Phase aktiviert werden? Begründen Sie Ihre Antwort.

d) Sowohl auf dem linken, als auch dem rechten Operanden-Bus existiert die Möglichkeit eine 0 auf den Bus zu schreiben. Nennen Sie einen Befehl für den bei der Ausführung eine 0 auf den linken Operanden-Bus geschrieben werden muss. Begründen Sie Ihre Antwort.

Lösung:

a) ACCLd, IRd, ALUAd, SPDd [falscher oder fehlender Treiber, -1]

b) IN2Dd, DDId [0.5 pro Treiber], nur diese Lösung ist richtig. Andere Lösungen mit 3 Treibern sind leider falsch.

c) Ja, man macht IN1Dd und DRd auf, dann steht der Inhalt von IN1 auf dem rechten Operandenbus. D.h. IN1Dd, DRd, ACCLd, ALUDId. [1 Antwort auf Frage, 0.5P pro richtigem Treiber], alternativ auch richtig: ACCDd, DRd, IN1LD, ALUDId.

d) `LOADI D i` da allein $0^{10}i$ in D geladen werden muss. (TODO) [0.5 Befehl, 0.5 Begründung]

Aufgabe 2 (4+2 Punkte)

In der Vorlesung haben Sie eine veränderte Form der aus TI bekannten ReTI kennengelernt, die um essentielle hardwaremäßige Grundlagen zur Implementierung eines Betriebssystems erweitert wurde. Unter anderem enthält die ReTI nun ein Register `SP` in dem der `Stack-Pointer` abgelegt wird sowie zwei Erweiterungen des `JUMP`-Befehls `INT i` und `RTI` zur Behandlung von Interrupts.

a) Nehmen Sie an, dass `SP` auf die erste Speicherzelle oberhalb des Stacks zeigt, d.h. wenn der aktuelle Stack in `M[n], ..., M[n+m]` abgelegt ist, dann gilt `<SP> = n-1` (und beim nächsten push soll dann `M[n-1]` verwendet werden). Diese Eigenschaft soll immer erhalten bleiben.
Geben Sie eine Implementierung unter Benutzung des erweiterten ReTI Befehlssatzes aus der Vorlesung für die Stack-Operationen `push()` und `pop()` an. Hierbei soll bei `push()` der Inhalt des Akkumulators `ACC` auf den Stack gelegt werden und bei `pop()` der oberste Eintrag des Stack in den Akkumulator geschrieben werden. *Gehen Sie davon aus, dass Sie den Stack beliebig befüllen bzw. leeren können und ignorieren Sie die Fehlerbehandlung (z.B. 'stack overflow').*

b) Warum mussten für die Interrupt-Behandlung die beiden atomaren Befehle `INT i` und `RTI` eingeführt werden? Weshalb können diese nicht einfach durch den restlichen Befehlssatz implementiert werden?

Lösung:

a) push:

```
STOREIN SP ACC 0
SUBI SP 1
```

pop:

```
LOADIN SP ACC 1
ADDI SP 1
```

oder auch

```
ADDI SP 1
LOADIN SP ACC 0
```

[2p pro Programm]

b) U.a. da der Program Counter weggeschrieben werden muss und bei der Wiederaufnahme nicht ins Programm, das `INT i` implementiert, zeigen soll. Bei `RTI` soll der Programmablauf am zurückgeladenen PC weitergeführt werden, es muss aber eigentlich z.B. noch der Stackpointer dekrementiert werden. [1p pro Befehl, mehrere Lösungen möglich]

Aufgabe 3 (6 Punkte)

Schreiben Sie ein C-Programm `test.c` mit folgendem Inhalt:

```c
#include <stdio.h>

int main(void)
{
   int value;
   FILE *fptr = fopen("myfile.txt","w");
   for(int i = 0; i < 2500; i++)
   {
      fprintf(fptr,"%d ",i);
   }
   fclose(fptr);
   fptr = fopen("myfile.txt", "r");
   for(int j = 0; j < 2500; j++)
   {
      fscanf(fptr, "%d ", &value);
   }
   fclose(fptr);
   return 42;
}
```

Bei den Funktionen `fopen, fclose, fprintf` und `fscanf` handelt es sich um Bibliotheksfunktionen aus `stdio`. Kompilieren und Linken Sie die Datei z.B. mit `gcc` per

```
$ gcc -o test test.c
$ chmod +x test
```

Machen Sie sich mit dem Befehl `strace` vertraut und führen Sie folgendes Kommando aus:

```
$ strace ./test
```

Interpretieren Sie die Ausgabe beginnend mit der Zeile mit dem Inhalt `"brk(0x ... )"`. Geben Sie an, welche Bibliotheksfunktion für die einzelnen Systemaufrufe verantwortlich sein könnten. Begründen Sie Ihre Antwort (Hinweis: Auch für Systemaufrufe sind `manpages` verfügbar. Achten Sie aber darauf, in einem geeigneten Manual, z.B. Linux Programmer's Manual, nach den Systemaufrufen zu schauen. Verschiedene Manuals zum selben Befehl lassen sich z.B. durch `man x Befehl`, wobei x die Nummer des Manuals ist, aufrufen.)

Tipps zur Ausführung: Sie können diese Aufgabe auf einer eigenen Maschine mit Linux-Betriebssystem, z.B. Ubuntu ausführen oder ein Linux-Betriebssystem über VirtualBox an Ihrem vorhandenen System einrichten. Hinweise und Links zu Ubuntu und VirtualBox finden Sie auf Übungsblatt 0. Alternativ kann die Aufgabe auch wie zuvor bei Übungsblatt 1 auf dem Login-Rechner der Universität (`ssh xy1234@login.uni-freiburg.de`) bearbeitet werden. Dafür können Sie entweder mit einem der vorhandenen Editoren (z.B. `vi`) die Datei `test.c` mit dem vorgegebenen Inhalt erstellen oder z.B. per `scp` die auf Ihrem Rechner lokale Datei `test.c` von Ihrem Rechner auf den Login-Rechner der Universität kopieren.

Lösung:

```
brk(0x2228000)                          = 0x2228000
/* alloziiert neuen Speicher (Datensegment des Prozesses) */
openat(AT_FDCWD, "myfile.txt", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
/* Von fopen erzeugt. Öffnet Datei mit relativem Pfad, andere Argumente: AT_FDCWD
legt fest, dass im current working directory geschaut werden soll.
Optionen Write-only, Create, wenn es die Datei noch nicht gab,
Datei leer machen. Mode für Permissions.*/
fstat(3, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
/* Fragt Dateistatus ab, file descriptor ist 3 (wie oben festgelegt),
Dateityp und -modus (mehr Info auf INODE manpage- nicht so wichtig),
regular file, st_size ist die Größe (bisher 0) */
write(3, "0 1 2 3 4 5 6 7 8 9 10 11 12 13 "..., 4096) = 4096
write(3, "041 1042 1043 1044 1045 1046 104"..., 4096) = 4096
write(3, "60 1861 1862 1863 1864 1865 1866"..., 3198) = 3198
/* Von fprintf erzeugt. Schreibt in File 3, den gegebenen String der Länge in Bytes.
Trotz 2500 fprintf Aufrufen nur 3 write Systemaufrufen, weil das Schreiben in die Datei gepuffert wird. */
close(3)                                = 0
/* Von fclose ausgelöst. Schließt den Dateizugriff von File 3.*/
openat(AT_FDCWD, "myfile.txt", O_RDONLY, 0666) = 3
/* Wie oben. Diesmal allerdings als Read Only Zugriff.  */
fstat(3, {st_mode=S_IFREG|0664, st_size=11390, ...}) = 0
/* Wie zuvor. Diesmal ist die Größe nicht mehr 0.
*/
read(3, "0 1 2 3 4 5 6 7 8 9 10 11 12 13 "..., 4096) = 4096
read(3, "041 1042 1043 1044 1045 1046 104"..., 4096) = 4096
read(3, "60 1861 1862 1863 1864 1865 1866"..., 4096) = 3198
read(3, "", 4096)                       = 0
/* Von fscanf ausgelöst. Liest Inhalt von File 3.*/
close(3)                                = 0
/* Von fclose ausgelöst. Schließt den Dateizugriff von File 3 wieder.*/
exit_group(42)                           = ?
/* Verlässt den Thread mit Return Code 42 */
+++ exited with 42 +++
```

[0.5p pro korrekter Befehlserklärung, jeweils 0.5p wenn bei `openat(...)` bzw. `open(...)` jeweils der Modus (Write Only und Read Only) korrekt erklärt oder zumindest erwähnt wurde]

Abgabe: als PDF im Übungsportal bis 04.11.2022 um 12:00