64-041 Übung Rechnerstrukturen und Betriebssysteme

Aufgabenblatt 10 Ausgabe: 14.12.2025, Abgabe 07.01.2026 24:00

Gruppe	
Name(n)	Matrikelnummer(n)

Aufgabe 10.1 (Punkte 5+5+5)

Lebensdauer von SSDs. Wegen der viel schnelleren Zugriffszeiten setzen sich Flash-Speicher (Solid State Disks, SSDs) zunehmend gegenüber magnetischen Festplatten durch. Ein Nachteil von SSDs bleibt die begrenzte Anzahl von Schreibzyklen, weil die extrem dünnen Isolierschichten der Floating-Gates in den Zellen beim Schreiben beschädigt werden.

Ein bekannter Hersteller garantiert für eine SSD mit 1.0 TByte Gesamtkapazität eine Lebensdauer von drei Jahren beziehungsweise eine maximale Anzahl von 400 TBW („400 Terabytes total bytes written“). Die SSD liefert beim sequentiellen Lesen maximal 3.2 GB/s an Daten, und kann laut Datenblatt mit 1900 MB/s (sequentiell) bzw. 600 MB/s (zufällige Zugriffe) beschrieben werden.

(a) Wie lange hält die SSD durch, wenn sie konstant mit der maximalen Schreibrate beschrieben wird (z.B. Videoüberwachung) ?

(b) Wie lange lebt die SSD, wenn dauerhaft Schreibzugriffe auf zufällige Sektoren erfolgen?

(c) Wir nehmen an, dass bei einem Laptop durchschnittlich etwa 20 GB Daten pro Tag auf die SSD geschrieben werden. Wie lange hält die SSD in dieser Betriebsart durch?

Aufgabe 10.2 (Punkte 5+5+5+5+10)

Datenblatt Lesen (Impulsdiagramme). Anders als Lehrbücher verzichten Datenblätter auf leicht verständliche Erklärungen und präsentieren statt dessen die relevante Information in möglichst kompakter Form. Als Beispiel betrachten wir das Datenblatt der integrierte Schaltung FAN5622 zur Ansteuerung von zwei LEDs mit digital einstellbarer Helligkeit. Die Leuchtdioden werden wie in Figure 1 gezeigt an das IC angeschlossen, und anschließend kann die Stromstärke an den Pins 4 und 6 (und damit die Helligkeit von LED1 und LED2) über digitale Steuerimpulse am Eingang `CTRL` geregelt werden, siehe `http://tams.informatik.uni-hamburg.de/lectures/2025ws/vorlesung/rsb/uebung/onsemi-FAN5626.pdf`.

Geben Sie für alle der folgenden Fragen jeweils an, wo (Abschnitt/Diagramm/Seite) Sie die gesuchte Information im Datenblatt gefunden haben, bzw. erläutern Sie Ihre Antworten:

(a) Ermitteln Sie zunächst die zulässige Betriebsspannung `VIN` des ICs.

(b) Wie kann die maximale Stromstärke der LEDs (und dazu proportional die gewünschte Helligkeit) auf 10 mA eingestellt werden?

(c) Welche Spannungswerte sind am `CTRL`-Eingang notwendig, um sicher als logisch 1 (`HIGH`) und logisch-0 (`LOW`) interpretiert zu werden?

(d) Wie lange muss das `CTRL`-Signal laut Tabelle auf `LOW` gehalten werden, um die LED abzuschalten? Wie lange dauert es tatsächlich laut Figure 15?

Hinweis: bei den gezeigten Impulsdiagrammen sind die x-Achse (Zeit) und y-Achsen (Spannungen/Ströme) jeweils in `Div` angegeben, wobei ein `Div` einem Kästchen der gepunkteten Hilfslinien entspricht. Beispielweise wechselt die Eingangsspannung `CTRL (2V/Div)` in Abbildung 15 zum Zeitpunkt $t = 200\,\mu$sec (zwei Kästchen nach rechts mit 100μs pro Kästchen) von 2 Volt (1 Kästchen hoch) auf nahe 0 Volt. Der Strom durch die LED wird dann nach knapp 8 Kästchen abgeschaltet (`ILED`) und gleichzeitig fällt natürlich auch die Stromaufnahme `I_IN` des ICs.

(e) Die Helligkeit der LEDs wird über ein Digitalsignal am Eingang `CTRL` geregelt, siehe Figure 16, Figure 17, und Tabelle 2.

Wir wollen ein Blinkmuster erzeugen, bei dem die LED zunächst auf 50% der maximalen Helligkeit eingestellt wird, und dann nach 1.0 sec auf die volle Helligkeit. Beschreiben Sie die dazu nötigen Signale am $CTRL$-Eingang und zeichnen Sie das Impulsdiagramm.

Aufgabe 10.3 (Punkte 5+5+5+10+15)

Akkumulator-Maschine. Die am Anfang der Vorlesung vorgestellte „Primitive Maschine“ verkörpert einen Digitalrechner der ersten Generation (ca. 1950). Es gibt nur ein Resultatregister (den „Akkumulator“) für die arithmetischen und logischen Operationen, und alle Datenoperationen und Sprungbefehle arbeiten mit absoluten Addressen. Es ist zwar unbequem, aber auch sehr lehrreich, ein paar Beispielprogramme für einen derartigen Rechner zu erstellen.

Die PRIMA verfügt über einen Hauptspeicher von 256 Bytes (Adressen 0..255) und insgesamt vier 8-bit Register: den Programmzähler (program counter, PC), das Befehlsregister (BR), ein Adressregister (AR), und den Akkumulator (ACCU). Der Befehlszyklus umfasst drei Takte: Befehl holen , Adresse holen, und Ausführung:

```
PRIMA Befehlszyklus:
   1. Holen:        BR := MEM[ PC }, PC := PC + 1
   2. Adresse:      AR := MEM[ PC ], PC := PC + 1
   3. Ausführung:   siehe Befehlssatz
```

Die Befehlsausführung startet nach einem Reset bei Speicheradresse 0. Als Beispiel für minimale Input/Output-Fähigkeiten gibt es zusätzlich einen von außen bedienten Schalter (switch, SW), der mit einem bedingten Sprungbefehl abgefragt werden kann.

```
PRIMA Befehlssatz:
   Opcode     Mnemonic    Bedeutung

   0x00       nop         no-operation
   0x01       clear       accu = 0
   0x02       load        accu = MEM[ AR ]
   0x03       store       MEM[ AR ] = accu

   0x10       incr        accu = (accu + 1) & 0xff
   0x11       decr        accu = (accu - 1) & 0xff
   0x12       add         accu = (accu + MEM[AR]) & 0xff
   0x13       sub         accu = (accu - MEM[AR]) & 0xff

   0x20       neg         accu = ~accu & 0xff     (bitwise not)
   0x21       and         accu = accu & MEM[AR]   (bitwise and)
   0x22       or          accu = accu | MEM[AR]   (bitwise or)
   0x23       xor         accu = accu ^ MEM[AR]   (bitwise xor)

   0x40       jump        PC = AR           (unconditional jump)
   0x41       bz          if (accu == 0) PC = AR    (branch if zero)
   0x42       bneg        if (accu & 0x80) PC = AR  (branch if negative)
   0x45       bsw         if (SW) PC = AR           (branch if switch)

   0xff       halt        stop machine execution
```

Sie können die folgenden Aufgaben gerne mit Papier und Bleistift lösen, aber natürlich ist die Verwendung einer virtuellen Maschine zu empfehlen. Wie stellen ein Python-Skript zur Verfügung, dass die PRIMA simuliert: href://tams.informatik.uni-hamburg.de/lectures/2025ws/vorlesung/rsb/uebung/prima.py.

Die Simulation ist text-basiert, kann also auch remote im Terminal ausgeführt werden, etwa (Remote/SSH-Zugang) auf `rzssh1.informatik.uni-hamburg.de`:

```
ssh <username> rzssh1.informatik.uni-hamburg.de
wget tams.informatik.uni-hamburg.de/lectures/2025ws/vorlesung/rsb/doc/prima.py
python prima.py
```

Im Python-Interpreter kann dann interaktiv herumgespielt werden, aber natürlich können Sie auch Ihre eigenen Skripte schreiben und starten:

```
prima = PRIMA()
prima.memset( 0, [42,11,2,3,4,5,6,7,8,9] ) # MEM[0] = 42, MEM[1] = 11, ...
prima.print_memory( 0, 127, 1 )             # MEM[0..127] list of hex-values
prima.print_registers()
prima.load_memory( "/tmp/demo.txt" )        # addr: hex hex hex hex ... \n
prima.clk()                                 # clock pulse (phase 1 or 2 or 3)
prima.instruction()                         # one instruction (phase 1+2+3)
prima.set_SW( 0 )                           # set input switch to 0 (or 1)
for i in range( 1000 ):
    prima.instruction( print_registers=True )
    if prima.is_halted(): break             # machine stopped
```

```
print.print_memory( 0, 256, 2 )            # MEM[0..255] addr + hex-values
prima.save_memory( "/tmp/factorial.txt", start=0, end=256,
                   comment="Prima factorial by A.Author" )
```

(a) Offenbar gibt es für die PRIMA keinen separaten „Load Immediate“ Befehl, um einen direkt im Befehl kodierten Zahlenwert in den Akkumulator zu laden. Überlegen Sie sich, einen Befehl (oder eine Befehlsfolge), startend ab Speicheradresse `0x30`, die den Wert `0x42` in den Akkumulator lädt.

(b) Schreiben Sie ein Programm, dass den Wert in der Speicherstelle `0xfc` zunächst einmal auf den Wert `0x03` initialisiert und anschließend in einer Endlosschleife immer weiter hochzählt.

Bitte geben Sie für diese und die folgenden Aufgaben das Programm jeweils als „ausführbare“ Textdatei (erzeugt mittels `prima.save_memory()`) und als kommentiertes Listing ab, wo sie die verwendeten Speicheradressen und jeden einzelnen Maschinenbefehl beschreiben.

(c) Erweitern Sie das Programm aus der vorherigen Ausgabe, so dass der Wert an der Speicherstelle `0xfc` abhängig vom Wert des Schalters `SW` in der Endlosschleife inkrementiert (SW=0) oder dekrementiert (SW=1) wird. Nutzen Sie zum Texten die `set_SW()`-Funktion des Simulators.

(d) Schreiben Sie ein Programm, dass die Werte an den Speicheradressen 248 und 249 (bzw. `0xf8` und `0xf9`) miteinander multipliziert und dann stoppt. Dabei soll das Ergebnis an die Adresse `0xfb` geschrieben werden. Um den Aufwand gering zu halten, berechnen wir nur das niederwertige Byte (also die Bits 7:0) des vollständigen 16-bit Resultats.

Da die PRIMA nicht über einen Multiplikationsbefehl verfügt, ist vermutlich eine kleine Schleife notwendig. Eine Möglichkeit ist es, den ersten Operanden so oft mit sich selbst zu addieren, wie der zweite Operand angibt.

(e) Erstellen Sie ein Programm zur Berechnung der Fakultät $n!$, wobei Sie den Code aus der vorherigen Aufgabe natürlich wiederverwenden können. Das Argument n soll dabei aus Speicheradresse `0xfc` gelesen und das Resultat (wiederum nur die unteren 8-Bits) in Adresse `0xff` abgespeichert werden.

Welche Werte berechnet Ihr Programm für die Argumente $n = 0$, $n = 5$, und $n = 7$?

Aufgabe 10.4 (Punkte 10+5)

Installation und Test eines C-Compilers (GNU Toolchain): In Vorbereitung auf Kapitel 13 zum x86-Assembler und analog zu den Beispielen in Kapitel 2 ab Folie 95, sollen Sie selbst Zugang zu einem C-Compiler und den zugehörigen Tools haben. Wir empfehlen die *GNU Toolchain* mit dem `gcc` C-Compiler und Werkzeugen. Die Programme sind auf Linux-Systemen in der Regel vorinstalliert, so dass Sie die Befehle direkt ausführen können.

Im Informatikum können Sie die Dual-Boot Rechner in den Poolräumen unter Linux nutzen. Wenn Sie zu Hause lernen und keinen Linux PC haben, bzw. die Cygwin-Umgebung (s.u.) nicht installieren wollen, können sie auch „Remote“ auf den Informatik Rechnern arbeiten: aus einem Linux-Terminal / der Windows-Eingabeaufforderung einloggen:

```
ssh rzssh1.informatik.uni-hamburg.de
```

. . . dort dann Start der Programme. **Achtung**: ihr Linux HOME-Verzeichnis ist `infhome`.

Für Windows-Systeme könnten Sie die Cygwin-Umgebung von cygwin.com herunterladen und installieren. Im Setup von Cygwin dann bitte den gcc-Compiler und die Entwickler-Tools auswählen und installieren. Natürlich können Sie auch jeden anderen C-Compiler verwenden, müssen sich dann aber die benötigten Befehle und Optionen selbst heraussuchen.

Anmerkung: keine Angst, die Aufgabe soll zeigen, wie Assemblercode aussieht und Ihnen helfen erste Einblicke zu gewinnen, wie Betriebssystem, (Programm-) Binär-Code und die Hardware zusammenspielen. Hauptsächlich soll diese Aufgabe dazu motivieren, auf dem eigenen Rechner mit den Werkzeugen zu „spielen“ und ein Gefühl dafür zu bekommen, wie der programmierte Code auf den niedrigeren Abstraktionsebenen und im Speicher des Rechners schließlich aussieht. **Es geht nicht darum Assemblerprogrammierung zu lernen!**

Für einen ersten Test tippen Sie bitte das folgenden Programm ab oder laden Sie sich die Datei `aufg10_4.c` herunter. Passen Sie die Datei an, indem Sie dort ihren Namen und die Matrikelnummer eintragen. Anschließend sollen Sie das Programm übersetzen und den dabei erzeugten Assembler- und Objektcode studieren.

```c
/* aufg10_4.c
 * Einfaches Programm zum Test des C-Compilers und der zugehörigen Tools.
 * Bitte setzen Sie in das Programm ihren Namen und die Matrikelnummer ein
 * und probieren Sie alle der folgenden Operationen aus:
 *
 * Funktion         Befehl                                    erzeugt
 * ----------------+-----------------------------------------+----------------
 * C -> Assembler:  gcc -S aufg10_4.c                 -> aufg10_4.s
 * C -> Objektcode: gcc -c aufg10_4.c                 -> aufg10_4.o
 * C -> Programm:   gcc -o aufg10_4.exe aufg10_4.c    -> aufg10_4.exe
 * Disassembler:    objdump -d aufg10_4.o
 *                  objdump -d aufg10_4.exe
 * Ausführen:       aufg10_4.exe
 */

#include <stdio.h>

int main( int argc, char** argv )
{ int matrikelNr    = 345678;
  char vorname[32]  = "Studi";
  char nachname[32] = "Informaticus";
//char *vorname     = "Studi";
//char *nachname    = "Informaticus";

  printf("Name: %s %s - Matrikelnr.: %d\n", vorname, nachname, matrikelNr);
  return 0;
}
```

(a) Machen Sie sich mit dem Compiler und den Tools vertraut. Probieren Sie die vorgeschlagenen Befehle aus und sehen Sie sich die Ausgaben an.

Erzeugen Sie eine Textdatei, die die Ausgabe des Programms und ein Listing des Disassemblers enthält. Dies geschieht am einfachsten mit den folgenden Befehlen:

```
./aufg10_4.exe          > loesung10_4.txt
echo "=========================================" >> loesung10_4.txt
objdump -d aufg10_4.o >> loesung10_4.txt
```

Markieren Sie in der Datei an welcher Stelle des Codes: Vorname, Nachname und Matrikelnummer stehen (mit kurzer Begründung). Diese Datei ist als Lösung des Aufgabenteils abzugeben.

(b) In dem Code `aufg10_4.c` sind die Zeilen 22 und 23 auskommentiert. Ändern Sie die Variablen für Vor- und Nachnamen in die zweite Version (Zeiger auf den String, statt `char`-Array).

Was ändert sich in dem Assembler-Code? Es genügt, die Änderungen (inhaltlich) zu beschreiben, es müssen keine Listings abgegeben werden.