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

Freiburg, 04.11.2022

Betriebssysteme
Musterlösung zu Übungsblatt 3

Aufgabe 1 (14 Punkte)

In der Vorlesung haben Sie eine UART als Beispiel für ein I/O-Gerät kennengelernt. Zur Erinnerung:

- Die UART hat 8 Register mit jeweils 8 Bit.
- Am Datenausgang der UART (verbunden mit dem Datenbus), werden die 8-Bit Worte mit 24 Nullen zu 32-Bit Worten aufgefüllt.
- Wenn eine Adresse $(a_{31}, a_{30}, ..., a_0)$ mit 01 beginnt, dann wird die UART angesprochen (Memory Map).
- Die Register werden mit den drei niederwertigsten Bits (a_2, a_1, a_0) vom Adressbus adressiert.

Wir nehmen an, dass die acht Register R0 - R7 der UART aufsteigend adressiert sind, d.h. R0 wird adressiert mit $010^{27}000$, R1 mit $010^{27}001$, usw..

Wir gehen jetzt von folgendem **vereinfachten Kommunikationsprotokoll** aus:

Wir betrachten die Register R0, R1 und R2 näher. Hierbei sei (aus CPU/ReTI-Sicht) R0 zum *Senden* von Daten an das Peripheriegerät, R1 zum *Empfangen* und R2 zum Lesen bzw. Schreiben von *Statusmeldungen*. In R2 hat jedes der acht Bit eine Statusfunktion. Der Einfachheit halber nehmen wir an, dass nur die beiden niederwertigsten Bits für uns relevant sind. Das niederwertigste Bit b0 von R2 wird wie folgt verwendet:

- b0 = 0 wenn das Senderegister (R0) von der CPU vollgeschrieben wurde. Die UART akzeptiert keine weiteren Daten mehr bis die aktuellen versendet wurden.
- b0 = 1 wenn das Senderegister von der CPU befüllt werden darf.

Das darauffolgende Bit b1 von R2 wird folgendermaßen verwendet:

- b1 = 0 wenn keine Daten im Empfangsregister (R1) sind bzw. diese schon von der CPU gelesen wurden.

- b1 = 1 wenn alle Daten von der Peripherie kommend im Empfangsregister (R1) bereit stehen.

Versenden: Wir nehmen an, dass beim Versenden von Daten von der CPU an das Peripheriegerät die CPU zuerst prüft, ob b0 = 1. Falls b0 noch 0 ist, ist die UART mit dem bitweisen Versenden der Daten in R0 beschäftigt und die CPU darf keine neuen Daten in R0 schreiben. Wenn b0 = 1, dann kann die CPU R0 mit neuen Daten beschreiben und setzt dann b0 auf 0. Dadurch wird die UART veranlasst, die Daten bitweise zu versenden und quittiert das Versenden durch Setzen von b0 auf 1.

Empfangen: Umgekehrt wartet beim Empfangen von Daten die CPU bis das Register R1 mit Daten gefüllt wurde. Die UART zeigt dies durch Setzen des Bits b1 = 1 an. In diesem Fall liest die CPU die Daten von R1 und setzt b1 = 0. Dadurch wird es der UART ermöglicht, wieder empfangene Daten im Empfangsregister R1 abzuspeichern. Nachdem die UART 8 weitere Bit in R1 geschrieben hat, setzt sie b1 wiederum auf 1.

Wir möchten nun ein ReTI-Programm schreiben, dass auf dem EPROM ausgeführt wird und über die UART ein Benutzerprogramm einliest. Dieses Benutzerprogramm (kleiner als $2^{16} - a$ Befehle) soll anschließend in den SRAM (Hauptspeicher) ab Adresse a abgelegt werden.

Die UART erhält ihrerseits die Daten bitweise von einem Peripheriegerät, das beispielsweise mit einem anderen Rechner verbunden ist, der die Befehle des Benutzerprogramms liefert. Wir nehmen an, dass im EPROM ab Adresse 0 schon ein Programmstück steht, das die Kommunikation mit diesem Peripheriegerät (dem anderen Rechner) initiiert, dabei u.a. die Übertragungsgeschwindigkeit aushandelt etc.. Unser Programm folgt nach dem Ende dieses Programmstücks ab Adresse m.

In Abb. 1 ist die Architektur schematisch dargestellt.

***Hinweis:* Bitte kommentieren Sie Ihren Code ausführlich!**

a) Schreiben Sie zunächst ein ReTI-Programm, dass in einer Schleife läuft und ständig überprüft ob b1 im Statusregister R2 auf 1 gesetzt wurde, d.h. ob der CPU vom Peripheriegerät Daten in R1 bereitgestellt wurden. Wenn Daten (8-Bit) bereitgestellt wurden, soll ihr Programm diese in das Register IN1 der ReTI laden und b1 wieder auf 0 setzen (sodass die UART wieder neue Daten in R1 schreiben darf). Beachten Sie, dass Sie vor Zugriffen auf die UART im Datensegmentregister DS das entsprechende Präfix 01 benötigen. Sie können annehmen, dass im EPROM (Präfix 00) an Adresse r die Konstante 010^{30} liegt. Beginnen Sie Ihr Programm also mit folgenden Zeilen:

```
LOADI IN1 0   // IN1 auf 0 setzen (hier kann spaeter Inhalt aus R1 addiert werden).
LOADI DS 0    // Zugriff auf Daten im EPROM
LOAD DS r     // Konstante 010...0 in DS laden --> Zugriff auf UART
LOAD ACC 2    // Statusregister R2 in Akkumulator laden.
  .           // ab hier folgt Ihr Programm.
  .
  .
```

b) Nun ist ihr Programm dazu in der Lage Daten abzuholen. Die Daten liegen allerdings als 8 Bit Worte vor. Ein ReTI Befehl ist aber 32 Bit breit, deshalb müssen sie einen Befehl aus vier 8-Bit Worten kombinieren. Lassen sie ihre Schleife aus a) immer viermal laufen und schieben

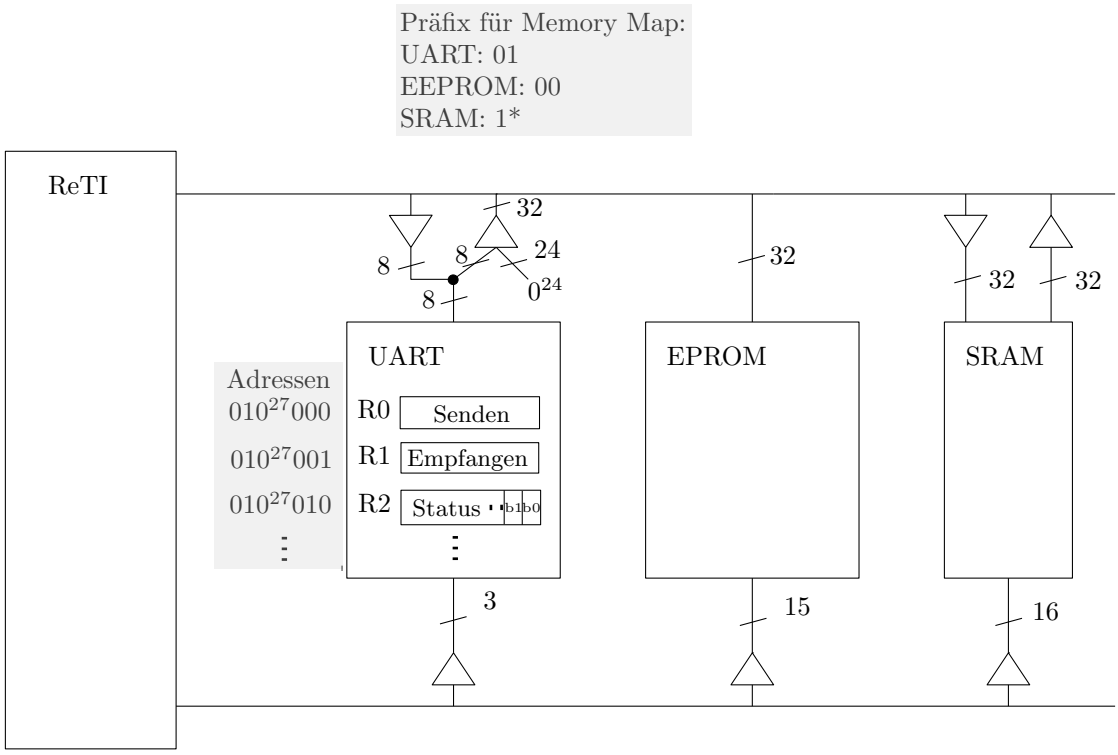

Abbildung 1: Architektur I/0.

sie den aktuellen Inhalt von IN1 (das sie mit den Daten aus R1 befüllen) bei jedem Abholen von Daten aus R1 um acht Stellen nach links (Linksshift). Den Programmcode aus a) dürfen Sie mit `POLLING-LOOP` oder ähnlichen Abkürzungen referenzieren.

```
LOADI IN2 4          // Benutze IN2 als Schleifenzaehler
POLLING-LOOP         // Code aus Teil a)
  .
  .
```

c) Nun kann Ihr Programm einen 32-Bit Befehl im Register IN1 abspeichern. Erweitern sie das Programm nun, sodass es den aktuellen Befehl aus IN1 an der entsprechenden Stelle im Hauptspeicher abspeichert (beginnend ab Adresse a). Beachten Sie hierbei, dass sie im Datensegmentregister DS das ensprechende Präfix benötigen (1). Sie können annehmen, dass im EPROM (Präfix 00) an Adresse s die Konstante 10^{31} liegt. Ihr Programm soll dazu in der Lage sein, alle obigen Schritte aus Teil a) und b) zu wiederholen bis schließlich von der UART der letzte Programmbefehl `LOADI PC 0` gelesen wird. Sie können zur Abfrage auf das Ende des Programms annehmen, dass im EPROM an der Adresse t die Konstante 01110000000...0 (Kodierung des Befehls `LOADI PC 0`) liegt. Nach der Übertragung des kompletten Programms springen Sie einfach an Speicherstelle a, ab der Sie das Benutzerprogramm abgelegt haben. Um Code, der die Kommunikation mit dem Peripheriegerät evtl. beendet, müssen Sie sich nicht kümmern.

(Der Einfachheit halber nehmen wir an, dass jedes Benutzerprogramm mit dem Befehl `LOADI PC 0` endet, so dass nach Abarbeiten des Programms wieder in Ihr EPROM-Programm gesprungen wird, das das nächste Benutzerprogramm über die UART lädt.)"

Lösung:

Lösungsskizze (grobe Idee): Polling-Schleife, die wartet bis das R2 (Status-Register) Bit b1 auf 1 gesetzt ist, das anzeigt, dass das Empfangsregister nun Daten enthält. Inhalt des Empfangsregisters in ein Register der ReTI laden. 8-Bit Teile von Befehlen kommen von der UART mit 0 aufgefüllt. Einen Zähler zyklisch von 3 bis 0 laufen lassen, Datum mit 2^8 multiplizieren und neues Datum drauf addieren.

Eigentlich nicht sinnvoll zu lösen: SRAM Adresse an die der Code geschrieben wird inkrementieren... Wir gehen einfach davon aus, dass diese aktuelle Adresse im EPROM an ebenfalls Adresse a steht. Eigentlich müsste man in einem zusätzlichen Register mitzählen oder so...

Zu den Punkten: Grobe Aufteilung ist 4P für a), jeweils 5P für b) und c). Die richtige Lösung muss natürlich nicht exakt so aussehen wie hier und allgemein könnt ihr die Bepunktung frei und großzügig entscheiden und kleine Punktabzüge für Fehler vergeben.

```
LOADI IN2 4           // Benutze IN2 als Schleifenzaehler.
LOADI IN1 0           // IN1 auf 0 setzen (hier kann spaeter Inhalt aus R1 drauf addiert werden).
/*--- Befehl-Auffuellen-Schleife*/
/*--- LSR Polling-Schleife ---*/
LOADI DS 0            // Zugriff auf Daten im EPROM.
LOAD DS r             // Lade Adresse r des EPROM mit Inhalt (010...0) in DS fuer UART-Zugriff.
LOAD ACC 2            // R2 (Adresse 2 in UART) in Akku laden.
ANDI ACC 2               // Berechne R2 && 0b10 (2). Wenn b1=1, steht im ACC jetzt eine 2, ansonsten 0.
JUMP_010 -2              // JUMP EQUAL. Wenn Bit 1 (b1) von R2 immernoch 0 -> Polling Schleife Neustart.

/*****************************/
MULTI IN1 256            // *2^8, Linksshift um 8 Stellen. Bei COMPUTE IMMEDIATE ist DS fuer Segment egal.
ADD IN1 1                // R1 Inhalt auf bestehenden Inhalt in IN1 addieren.
                      // Fuer COMPUTE MEMORY muss richtiges Segment in DS gesetzt sein.
LOADI ACC 1..101            // Lade ACC mit (1...101) bzw. 2^32 - 3.
AND ACC 2                // Berechne ACC && R2. Ergebnis soll in R2 kommen.
STORE ACC 2              // Dadurch wird b1=0 gesetzt. Andere Bits bleiben erhalten.
SUBI IN2 1               // Schleifenzaehler dekrementieren.
MOVE IN2 ACC             // Schleifenzaehler aus IN2 in Akku laden.
JUMP_101 -10             // JUMP NOT EQUAL. Wenn Schleifenzaehler nicht 0, hole naechste 8 Bit von UART.
(ALTERNATIV: JUMP_010 2 )     // Schleifenzaehler ist = 0, d.h. in IN1 steht ein 32-Bit Befehl -> raus.
(ALTERNATIV: JUMP_111 -11 )   // JUMP Befehl-Auffuellen-Schleife Neustart

/*****************************/
LOADI DS 0               // DS wieder auf 0 setzen um aus EPROM zu laden.
LOAD ACC a               // Annahme: a enthaelt Adresse an der der letzte Befehl in SRAM
                         //geschrieben wurde
LOAD DS s                // Lade Adresse s des EPROM mit Inhalt (10...0) in DS fuer SRAM-Zugriff.
STOREIN ACC IN1 1        // Befehl aus IN1 in Zelle a+1 des SRAM abspeichern.
MOVE IN1 ACC             // Befehl in Akku schieben, Test, ob LOADI PC 0 Befehl (Ende der Uebertragung).
LOADI DS 0               // DS wieder auf 0 setzen um aus EPROM zu laden.
OPLUS ACC t              // Kodierung fuer LOADI PC 0 Befehl steht in t.
                         // ACC = 0, genau dann wenn in IN1 LOADI PC 0 stand.
JUMP_101 (noch zu zaehlen ...)  // JUMP NOT EQUAL. Wenn ACC != 0, ist Uebertragung nicht beendet.
                         // Springe zurueck zum Anfang des Programms.
LOAD ACC s               // Lade ACC mit (10...0) fuer SRAM Mapping.
ORI ACC a                // Berechne ACC || a -> ergibt Speicheradresse a im SRAM.
MOVE ACC PC              // Uebertragung beendet. Springe an Adresse a im SRAM wie in der Aufgabe vorgeschrieben.
```

Aufgabe 2 (6 Punkte)

In Aufgabe 1 haben Sie unter Verwendung von LOAD- und STORE-Befehlen abwechselnd mit einem Programm auf dem EPROM auf Adressen im Adressraum des UART (01) und des SRAM (1*) zugegriffen und daher das Datensegmentregister `DS` immer wieder neu belegen müssen (für die Ergänzung der 22-Bit Adressen mit einem Präfix).

Welche Befehle können Sie anstatt `LOAD` und `STORE` verwenden, um ohne diese abwechselnde Neubelegung von DS auszukommen? Begründen Sie Ihre Antwort.

Hinweis: Falls Sie für Ihre Implementierung große Konstanten verwenden wollen, dann können Sie annehmen, dass diese an geeigneten Speicherstellen im EPROM stehen und von dort geladen werden können.

Lösung:

Wir nehmen an, dass

```
LOAD IN1 a //an Adresse a ist 010...0 (32 Bit) gespeichert
LOAD IN2 b //an Adresse b ist 10...0 (32 Bit) gespeichert
```

dann kann man z.B. per

```
LOADIN IN1 ACC 1 //Adresse 1 im UART ansteuern
LOADIN IN2 ACC 1 //Adresse 1 im SRAM ansteuern
```

eine Adresse im UART / SRAM ansteuern.

Abgabe: als PDF im Übungsportal bis 11.11.2022 um 12:00