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 (verbunden über UART), 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.

Bitte führen Sie Ihren Code für die unten aufgeführten Arbeitsanweisungen mit dem RETI-Interpreter aus!

Folgendes Programm `program.reti`

```
INT 3 # just a breakpoint, use c in debug mode -d to directly jump here
INT 0
MOVE ACC IN1
INT 0
MULT ACC IN1
INT 1
JUMP 0
```

verwendet Ihre Interrupt Service Routinen. Benutzen Sie dieses, um Ihre Interrupt Service Routinen im RETI-Interpreter zu testen. Nutzen Sie hierzu folgenden Befehl:

```
$ reti_interpreter -i interrupt_service_routines.reti -d program.reti
```

Wir implementieren in dieser Aufgabe zwei Interrupt-Service-Routinen (ISR). Benutzen Sie für Ihr Programm `interrupt_service_routines.reti` folgendes Grundgerüst:

```
0000    IVTE 2 # INT 0: Get user input over UART and save it in ACC
0001    IVTE <i> # INT 1: Print an integer stored in ACC over UART
0002    SUBI SP 2 # start of ISR 0 at relative address 2 + 2^31
0003    STOREIN SP DS 1
0004    STOREIN SP IN1 2
....    # your reti code goes here
i-03    LOADIN SP DS 1
i-03    LOADIN SP IN1 2
i-02    ADDI SP 2
i-01    RTI
i+00    SUBI SP 4 # start of ISR 1 at address <i> + 2^31
i+01    STOREIN SP DS 1
i+02    STOREIN SP IN1 2
i+03    STOREIN SP IN2 3
i+04    STOREIN SP ACC 4
i+05    LOADI DS 1048576
i+06    MULTI DS 1024
i+07    MOVE ACC IN2
i+08    LOADI ACC 4
i+09    STORE ACC 0
i+10    LOAD ACC 2
i+11    ANDI ACC -2
i+12    STORE ACC 2
i+13    LOAD ACC 2
i+14    ANDI ACC 1
i+15    JUMP== -2
i+16    # your reti code goes here
i+17    LOADIN SP DS 1
i+18    LOADIN SP IN1 2
i+19    LOADIN SP IN2 3
i+20    LOADIN SP ACC 4
i+21    ADDI SP 4
i+22    RTI
```

Mithilfe der Kommandozeilenoption `-i` ist der RETI-Interpreter in der Lage die RETI-Befehle für **Interrupt Service Routinen** aus einer Datei `interrupt_service_routines.reti` herauszulesen und an den Anfang des simulierten SRAM, vor das geladene Programm aus `program.reti` zu schreiben. Mithilfe von `INT i` kann wie in der Vorlesung erklärt an den Anfang jeder dieser Interrupt Service Routinen `i` gesprungen werden. Mittels `RTI` kann am Ende einer Interrupt Service Routine wieder an die nächste Stelle im ursprünglichen Programm zurückgesprungen werden, an der dieses mittels `INT i` unterbrochen wurde.

Mittels der Direktive `IVTE i` können **Einträge einer Interrupt Vector Tabelle** mit der korrekten Startadresse einer Interrupt Sevice Routine erstellt werden. Die SRAM Konstante wird automatisch auf `i` draufaddiert.

a) Schreiben Sie eine ISR, die eine Eingabe aus der UART empfängt und diese dann in den `ACC` schreibt. 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 über die UART Daten in R1 bereitgestellt wurden. Wenn Daten (8-Bit) bereitgestellt wurden, soll ihr Programm diese in den `ACC` der ReTI laden und b1 wieder auf 0 setzen (sodass die UART wieder neue Daten in R1 schreiben darf).

b) Schreiben Sie eine zweite ISR, die eine im `ACC` gespeicherte Ganzzahl über die UART versendet. Sie überprüfen also vor jedem Versenden, ob die UART bereit ist (b0 = 1) und das Senderegister befüllt werden darf. Dann befüllen Sie jeweils das Senderegister mit 8-Bit Portionen des Wortes in `ACC`.

   Beachten Sie, dass Sie vor Zugriffen auf die UART im Datensegmentregister `DS` das entsprechende Präfix 01 benötigen. Sie müssen also die 32-Bit Konstante 010^{30} mit der Methode aus Aufgabe 1) in das `DS` Register bringen.

   Anmerkung: Interrupt Service Routinen müssen die ursprünglichen Registerinhalte auf dem Stack abspeichern. Nach Beendigung und vor Aufruf von `RTI` müssen diese dann wiederhergestellt werden.

`interrupt_service_routines.reti`

```
IVTE 2 # INT 0: Get user input over UART and save it in ACC
IVTE 53 # INT 1: Print an integer stored in ACC over UART
SUBI SP 2 # start of ISR 0
STOREIN SP DS 1
STOREIN SP IN1 2
LOADI DS 1048576
MULTI DS 1024
LOAD ACC 2
ANDI ACC -3
STORE ACC 2
LOAD ACC 2
ANDI ACC 2
JUMP== -2
LOAD ACC 1
MOVE ACC IN1
ANDI ACC 128
JUMP== 4
LOADI ACC -1
OPLUSI ACC 255
OR IN1 ACC
MULTI IN1 1048576
MULTI IN1 16
LOAD ACC 2
```

```
ANDI ACC -3
STORE ACC 2
LOAD ACC 2
ANDI ACC 2
JUMP== -2
LOAD ACC 1
MULTI ACC 65536
OR IN1 ACC
LOAD ACC 2
ANDI ACC -3
STORE ACC 2
LOAD ACC 2
ANDI ACC 2
JUMP== -2
LOAD ACC 1
MULTI ACC 256
OR IN1 ACC
LOAD ACC 2
ANDI ACC -3
STORE ACC 2
LOAD ACC 2
ANDI ACC 2
JUMP== -2
LOAD ACC 1
OR IN1 ACC
MOVE IN1 ACC
LOADIN SP DS 1
LOADIN SP IN1 2
ADDI SP 2
RTI
SUBI SP 4 # start of ISR 1
STOREIN SP DS 1
STOREIN SP IN1 2
STOREIN SP IN2 3
STOREIN SP ACC 4
LOADI DS 1048576
MULTI DS 1024
MOVE ACC IN2
LOADI ACC 4
STORE ACC 0
LOAD ACC 2
ANDI ACC -2
STORE ACC 2
LOAD ACC 2
ANDI ACC 1
JUMP== -2
```

```
MOVE IN2 IN1
LOADI ACC 2097151
MULTI ACC 1024
ORI ACC 1023
AND IN1 ACC
DIVI IN1 256
DIVI IN1 65536
MOVE IN2 ACC
JUMP>= 2
ORI IN1 128
STORE IN1 0
LOAD ACC 2
ANDI ACC -2
STORE ACC 2
LOAD ACC 2
ANDI ACC 1
JUMP== -2
MOVE IN2 IN1
LOADI ACC 2097151
MULTI ACC 1024
ORI ACC 1023
AND IN1 ACC
DIVI IN1 65536
STORE IN1 0
LOAD ACC 2
ANDI ACC -2
STORE ACC 2
LOAD ACC 2
ANDI ACC 1
JUMP== -2
MOVE IN2 IN1
LOADI ACC 2097151
MULTI ACC 1024
ORI ACC 1023
AND IN1 ACC
DIVI IN1 256
STORE IN1 0
LOAD ACC 2
ANDI ACC -2
STORE ACC 2
LOAD ACC 2
ANDI ACC 1
JUMP== -2
STORE IN2 0
LOAD ACC 2
ANDI ACC -2
```

```
STORE ACC 2
LOAD ACC 2
ANDI ACC 1
JUMP== -2
LOADIN SP DS 1
LOADIN SP IN1 2
LOADIN SP IN2 3
LOADIN SP ACC 4
ADDI SP 4
RTI
```