Unbedingt vor Bearbeitung des Blattes lesen:

**Bitte** benutzen Sie für alle RETI-Aufgaben die **neuste Version** des RETI-Interpreters[1] von

`https://github.com/matthejue/RETI-interpreter`

Der RETI-Interpreter ist dazu in der Lage, die RETI-Befehle eines in einer `.reti`-Datei angebenen RETI-Programms zu interpretieren, d.h. er kann das RETI-Programm **ausführen**, indem er die RETI-Befehle aus einer Datei `programm.reti` herausliest und in den simulierten SRAM schreibt und mithilfe eines autogenerierten EPROM-Startprogramms, dass zu Beginn ausgeführt wird an den Start dieses Programms springt.

Mithilfe der Kommandozeilenoption `-d` ist der RETI-Interpreter ebenfalls in der Lage das Programm zu **debuggen**, d.h. er zeigt die Speicher- und Registerinhalte nach Ausführung eines jeden Befels an, zwischen denen der Benutzer sich mittels `n` und dann `Enter` bewegen kann. Wird `INT 3` in das RETI-Programm geschrieben stellt dies einen Breakpoint dar, zu dem mittels `c` und dann `Enter` gesprungen werden kann. In der **neusten Version** der RETI-Interpreters stehen alle Aktionen, die Sie in diesem Debug-Modus ausführen können in der untersten Zeile des Text-User-Interfaces (TUI).

Mithilfe der Kommandozeilenoption `-i` ist er 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 `programm.reti` zu schreiben. Mithilfe von `INT i` kann wie in der Vorlesung erklärt an den Anfang jeder dieser Interrupt Service Routinen `i` gesprungen werden. Mittles `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.

**Bitte** **installieren** Sie den RETI-Interpreter entsprechend der angepinnten Anleitung im Forum.

Des Weiteren **benutzen** Sie den RETI-Interpreter **auf diesem** Übungsblatt (für **Aufgabe 2**) folgendermaßen:

```
$ bin/reti_interpreter_main -i interrupt_service_routines.reti -d programm.reti
```

im Verzeichnis oder, falls Sie den RETI-Interpreter installiert haben, einfach per

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

Denken Sie daran, dass Ihr Programm `programm.reti` immer mit einem

```
JUMP 0
```

abgeschlossen sein muss.

**Bitte** geben Sie die RETI-Codeschnipsel, die ihrer Meinung nach die geforderten Aufgaben lösen in Dateien `programm.reti` und `interrupt_service_routines.reti` ab. `programm.reti` ist für den RETI-Code in Aufbabe a) und den RETI-Code, der `INT` ersetzen soll in Aufgabe b) vorgesehen. `interrupt_service_routines.reti` ist für den RETI-Code, der `RTI` ersetzen soll in Aufgabe b) vorgesehen. Separieren Sie die einzelnen Subaufgaben in `programm.reti` mithilfe von Zeilenkommentaren `# a)`, `# push)`, `# pop)`, `# INT Ersatz)`.
In `interrupt_service_routines.reti` können Sie den Beginn ihres RTI Ersetzungscodes mithilfe von `# RTI Ersatz)` kenntlich machen. Ihre Antwort auf die Frage in Aufgabe b) kann auch mithilfe mehrerer Zeilenkommentare in Ihrer Datei `programm.reti` angegeben werden oder in Ihrer PDF. Der Teil ihrer Lösung, den Sie in `programm.reti` einreichen sollen, sollten ungefähr wie folgt aussehen:

[1] Die aktuelle Version ist `v1.0.1`.

```
# a)
LOADI ACC 42
# push
RETI-Instructions...
# pop
RETI-Instructions...
# b)
# INT Ersatz
RETI-Instructions...
# Ihre Antwort auf die Frage in Aufgabe b)
# für die Sie mehrere Zeilenkommentare verwenden können
JUMP 0
```

Der Teil ihrer Lösung, den Sie in `interrupt_service_routines.reti` einreichen sollen, sollten ungefähr wie folgt aussehen:

```
IVTE 1
# b)
# Möglicher RETI-Code für eine Interrupt Service Routine (nicht gefordert)
RETI-Instructions...
# RTI Ersatz
RETI-Instructions...
```

**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) Implementieren Sie die beiden atomaren Befehle `INT i` und `RTI` mithilfe der anderen RETI-Befehle als Teil der Dateien `programm.reti` und `interrupt_service_routines.reti`. `INT i` soll als Teil der Datei `programm.reti` implementiert werden und `RTI` als Teil der Datei `interrupt_service_routines.reti`, wie es in dem Einführungstext am Anfang dieses Übungsblattes erklärt wurde. Was könnte ein Grund sein, warum in Prozessorarchitekturen für die Interrupt-Behandlung meist atomare Befehle, die ähnlich zu `INT i` und `RTI` sind eingeführt werden?