Sie werden unter Verwendung von LOAD- und STORE-Befehlen immer wieder abwechselnd mit einem Programm auf Adressen im Adressraum des EPROM (00), der UART (01) und des SRAM (1*) zugreifen müssen.

Daher muss das Datensegmentregister `DS` immer wieder neu belegt werden (für die Ergänzung der 22-Bit Adressen mit einem Präfix).

Beispiel UART: Offensichtlich ist es nicht möglich, einfach $w = 010...0$ (32 Stellen) per `LOADI DS i` in `DS` zu laden, da `i` nur 22 Stellen erlaubt.

Eine Lösung wäre beispielsweise, die obersten 22 Bit von

`0100000000000000000000``0000000000`

zu laden, per:

```
LOADI DS 1048576 # (01000000000000000000)
```

und dann um 10 Stellen nach links zu shiften (per Multiplikation mit 2^{10}):

```
MULTI DS 1024
```

Wir nehmen zudem an, dass nicht nur die obersten Bits interessant sind, sondern die gesamte 32-Bit Konstante. D.h. die unteren 10 Bit müssen noch nachgeladen werden (z.B. per `ORI`).

Arbeitsanweisung:

Schreiben Sie ein RETI-Programm `constants.reti`, welches das 32-Bit-Wort

$1431655765 = [01010101010101010101010101010101]_2$

im `ACC` speichert.

Bonus: Besondere Vorsicht ist geboten, wenn Konstanten (z.B. ein Befehl) geladen werden, die an der vordersten Bit-Stelle eine 1 haben. Beim Shift per Multiplikation mit einer positiven Zahl wird Zweier-Komplement angenommen und die vorderste Bit-Stelle wird immer 0 bleiben, da bei der Multiplikation mit einer positiven Zahl nicht plötzlich eine negative Zahl (das vorderste Bit auf 1 gesetzt) als Ergebnis rauskommen kann, sofern nicht gezielt undefiniertes Verhalten bei einem Overflow ausgenutzt wird.

Schreiben Sie als Bonus auf ihr RETI-Programm in `constants.reti` folgend und am besten durch einen Kommentar `# bonus` abgegrenzt ein Programm, welches das 32-Bit-Wort

$-1431655766 = [10101010101010101010101010101010]_2$

im `ACC` speichert.

Bitte testen Sie Ihr Programm mit dem **RETI-Interpreter**! Nutzen Sie hierzu den folgenden Befehl:

```
$ reti_interpreter -d constants.reti
```

Mithilfe der Kommandozeilenoption `-d` ist der RETI-Interpreter 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).

Tipp: Mit der Kommandozeilenoption `-b` werden alle Registerinhalte, Memoryinhalte und Immediates im Binärsystem angezeigt, damit lässt sich beim debuggen leichter das ganze Shiften nachvollziehen.

```
# output: 1431647027 -1431647028
INT 3 # just a breakpoint, use c in debug mode -d to directly jump here
# Konstante, bei der bit 31 auf 0 gesetzt ist
# Als Beispiel hier: 0b01010101_01010101_00110011_00110011
LOADI ACC 349525 # 0b01010101_01010101
MULTI ACC 65536
ORI ACC 13107 # 0b110011_00110011
INT 0
INT 3 # just a breakpoint, use c in debug mode -d to directly jump here
# Konstante, bei der bit 31 auf 1 gesetzt ist
# Als Beispiel hier: 0b10101010_10101010_11001100_11001100
# Das Geht nicht so einfach, weil man durch multiplizieren eine Zahl nicht
# plötzlich negativ (das MSB auf 1) bekommen kann ohne sich auf undefined
# behaviour verlassen zu müssen
LOADI ACC -21846 # 0b101010_10101010 - 2**15 oder 0b11111_10101010_10101010 - 2**21
MULTI ACC 65536
ORI ACC 52428 # 0b11001100_11001100
INT 0
JUMP 0
```