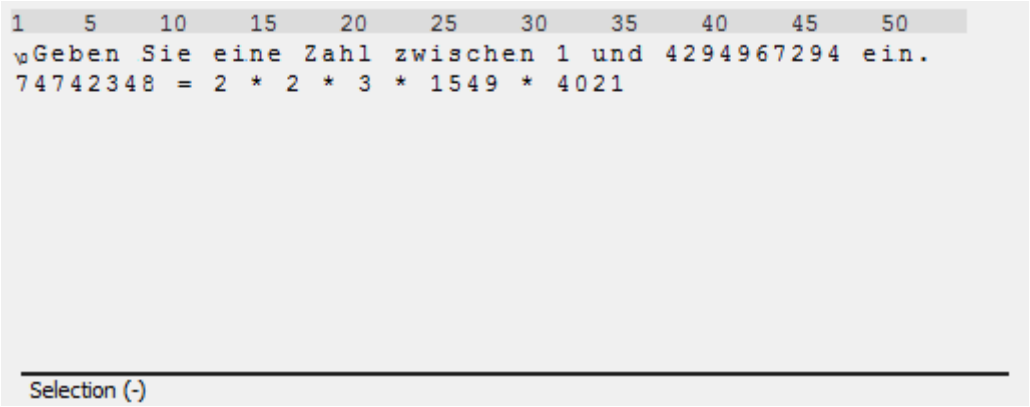
Berner Fachhochschule
Technik und Informatik

Software-Dokumentation

Primfaktorzerlegung mit ASM



Autoren: J. Haldemann, N. Käser, S. Plattner

Betreuer: Roger Weber

Abgabe: 20.01.2013

Abstract:

Dieses Dokument beschreibt die Entwicklung eines Software-Projektes im Rahmen des Moduls BTE5053b der Berner Fachhochschule für Technik und Informatik. Es dient als Referenz für den Betrieb der Software und ermöglicht die schnelle Einarbeitung zukünftiger Entwickler.

Vorwort

Im Rahmen des *Moduls BTE5053b: Hardwarenahe Softwareentwicklung*, entwickelten die Studierenden in kleinen Gruppen Softwareprojekte. Ziel dieser Projekte war es, die vorgängig im Unterricht erlangten Fähigkeiten im Bereich Hardwarenahe Softwareentwicklung in der Praxis anwenden zu können.

Die Studierenden konnten die zu entwickelnden Projekte in einem definierten Rahmen selbständig auswählen. Es musste sich um eine Aufgabe handeln, welche die Kommunikation über die RS232-Schnittstelle erforderlich macht. Der Schwerpunkt lag auf dem Anwenden der Hardwarenahen Programmiersprache Assembler. Als Zielhardware dient ein an der Schule entwickeltes sogenanntes Carme-Kit (carme.bfh.ch).

Wir entschieden uns zum Implementieren eines Algorithmus zur Primfaktor-Zerlegung auf dem Carme-Kit. Ein Computer soll dabei als Schnittstelle zwischen dem Benutzer und dem Algorithmus dienen. Computer und Carme-Kit sind dabei via RS232-Schnittstelle verbunden.

Pflichtenheft

Gemäss Absprache mit dem Dozenten wird folgendes Pflichtenheft umgesetzt:

Der eigentliche Algorithmus sowie der nötige Code zum Umgang mit dem UART (also der Kommunikation via RS232) werden in ASM implementiert, der restliche Code in C. Computerseitig wird eine Hyperterminal-Software von einem Drittanbieter eingesetzt.

Dem Programm sollen alle möglichen Strings mit einer Länge von maximal 50 Zeichen übergegeben werden können. Wenn es sich bei diesem String um eine positive Ganzzahl handelt, soll diese Zahl zerlegt werden und dem Benutzer ein String bestehend aus den Primfaktoren und nötigenfalls aus Trennzeichen zurückgegeben werden. Das Carme-Kit ist ein 32bit System. Deshalb ist der Definitionsbereich auf $0…2^{32}-1$ beschränkt. Um die Verarbeitung zu vereinfachen, wird dabei $2^{32}-1$ als „Fehler-Flag“ verwendet. Es ergibt sich also ein nutzbarer Definitionsbereich für unsere Funktion von $0…2^{32}-2$. Die Zahlen 0 und 1 werden – obwohl sie keine Primzahlen sind – als solche zurückgegeben. Das ist konsistent zu anderen Implementierungen der factor() Funktion. Exemplarisch sei hier auf die Matlab-Software verwiesen.

„743“	→	„743“
„124“	→	„2 * 2 * 31“
„13.2“	→	Fehlermeldung
„-41“	→	Fehlermeldung
„aes“	→	Fehlermeldung
„0“	→	„0“
„1“	→	„1“
$2^{32}-2$	→	Fehlermeldung

Während dem Zerlegen einer Zahl, nimmt das Programm keine weiteren Eingaben mehr an.

Inhalt

Bedienungsanleitung

Ziel einer Primfaktor-Zerlegung ist es, eine Zahl im Definitionsbereich in ihre Primfaktoren zu zerlegen. Es wird also die Menge aller Primzahlen bestimmt, deren Multiplikation zur Ursprünglichen Zahl zurück führt.

Aus der Definition der Primzahlen folgt für den Definitionsbereich des Algorithmus, dass nur ganze und nur positive Zahlen (und null) zerlegt werden können.

Der Algorithmus selbst weist keine obere Grenze für den Definitionsbereich auf. Aus der Architektur des Zielsystems ergibt sich aber sehr wohl eine obere Grenze. Wir wollen nur Zahlen zulassen, welche mit 32 Bit ausgedrückt werden können. So ergibt sich eine obere Grenze von 2^{32}-1.

Weiterhin haben wir uns entschieden, einen einzelnen Wert im Definitionsbereich zu reservieren, welcher den Fehlerfall signalisiert. Es drängt sich ein Wert am Rand des Definitionsbereichs auf. Den Wert null wollen wir als gültigen Wert betrachten und so entschieden wir uns für 2^{32}-1 als Fehler-Flag. Es ergibt sich also ein nutzbarer Bereich von 0…2^{32}-2.

Definitionsbereich: $0,1,2,\ldots,2^{32}-2$

Initialisieren der Hardware

Vor dem Benutzen soll die Hardware initialisiert werden. Dazu muss das Carme-Kit via RS232-Schnittstelle mit einem Computer verbunden und eingeschaltet werden. Auf dem Computer wird eine Hyperterminal-Sitzung[1] gestartet und mit der RS232-Schnittstelle verbunden. (Wir empfehlen, das mitgelieferte HTerm mit dem dazugehörigen Config-File zu verwenden. Die Einstellungen werden so automatisch vorgenommen)

Eingabe

Nach dem Starten der Hyperterminal-Sitzung ist das System bereit. Es können Zahlen als String (mit ASCII-Codierung) an das angeschlossene Carme-Kit gesendet werden. Das sollte (je nach verwendetem Hyperterminal-Programm) der Standardeinstellung für die Codierung entsprechen. Die Zahl 713 kann also einfach als 7 - 1 - 3 auf der Tastatur eingegeben werden. Mittels Enter-Taste wird die Eingabe beendet und die Software zum Auswerten veranlasst.

Eingaben, die gemacht werden während dem bereits eine Zahl zerlegt wird, werden verworfen. Es ist nicht möglich, eine laufende Rechnung zu unterbrechen (ausser via Hardware-Reset).

Berechnung

Nach Empfang des Enter-Signals (ausgelöst durch das Drücken der Enter-Taste) beginnt die Software mit der Bearbeitung der Eingabe. Bis heute liefert die Mathematik keine effiziente Möglichkeit zur Zerlegung einer Zahl in ihre Primfaktoren. Der Rechenaufwand hängt aber nicht wesentlich von der Grösse der Zahl ab, sondern von der Grösse und Anzahl ihrer Primfaktoren. Zahlen mit ausschliesslich kleinen Primfaktoren werden schnell zerlegt, Zahlen mit grossen Primfaktoren deutlich langsamer.

[1] Protokoll der RS232 Verbindung: Baud 9600, no Parity, 1 Stop-Bit, 8 Data-Bit

Normalerweise weiss man nicht im Voraus, wie gross die in einer Zahl enthaltenen Primfaktoren sind. Dadurch lässt sich eine Abschätzung der benötigten Rechenzeit nur für den schlechtesten Fall angeben. Der Rechenaufwand steigt dann etwa quadratisch mit der zu zerlegenden Zahl.

Der Rechenaufwand ist dann am grössten, wenn die grösste im Definitionsbereich enthaltene Zahl mit nur einem Primfaktor zerlegt werden soll. Das ist die Primzahl 4294967291.

Ausgabe

Nach der Berechnung werden die einzelnen Primfaktoren als String an das Hyperterminal-Programm auf dem Computer übermittelt. Die einzelnen Primfaktoren sind dabei durch die Zeichenfolge ‘ * ‘ (mit Leerschlägen vor und nach dem *-Zeichen) voneinander getrennt und aufsteigend sortiert. Die Antwort aus der Eingabe „713“ wäre also „23 * 31“.

Ausserdem wird im Falle einer ungültigen Eingabe ein String mit einer Fehlermeldung zurückgegeben.

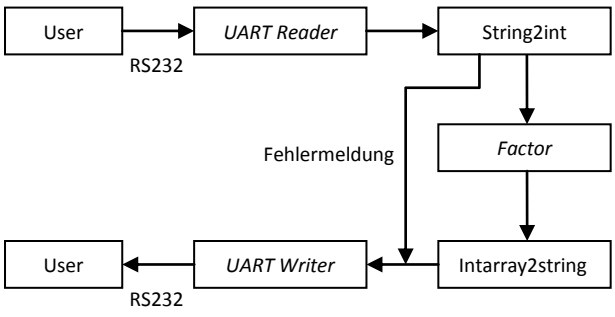
```
1    5    10    15    20    25    30    35    40    45    50
Geben Sie eine Zahl zwischen 1 und 4294967294 ein.
Keine Zahl = Ungueltige Eingabe

Selection (-)
```

Modulbeschreibung

Das Programm besteht aus verschiedenen Modulen. Diese verschiedenen Module sind teilweise einzeln, teilweise aber auch zusammen mit anderen Modulen in einer der Code-Dateien untergebracht. Im Nachfolgenden sind die Module beschrieben, ihre Anordnung entspricht nicht notwendigerweise der Anordnung innerhalb der verschiedenen Code-Dateien.

Nachfolgende Übersicht soll die Zusammenarbeit der Module veranschaulichen:



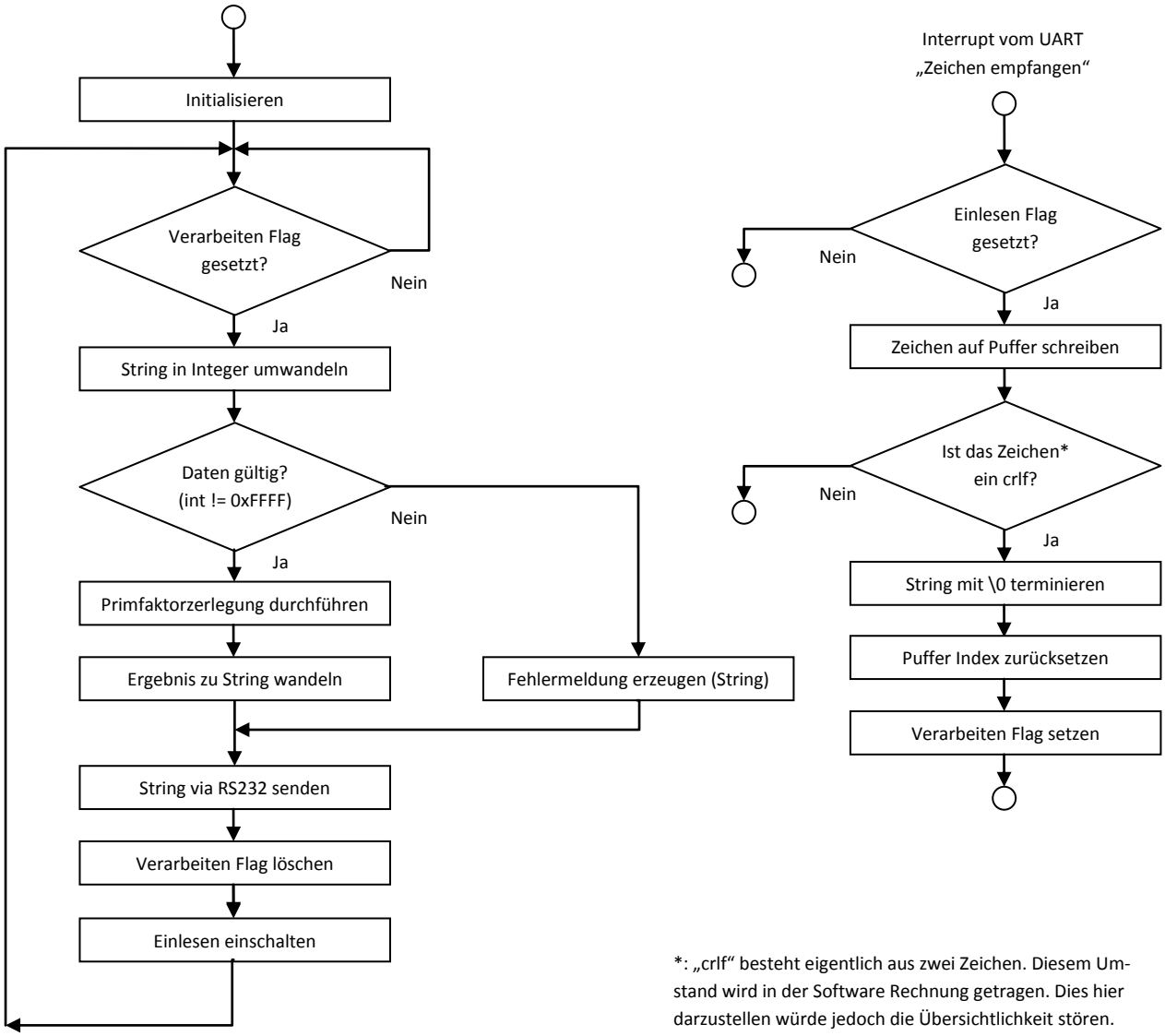
Die hier in *kursiver Schrift* dargestellten Module sind in ASM implementiert.

Design

Die Arbeitsweise des Programmes kann man sich als State-Maschine vorstellen:

- Nach dem Initialisieren liest das Programm solange die Daten am RS232-UART ein, bis ein Enter-Zeichen (also ein crlf) empfangen wird.
- Die Daten werden auf Gültigkeit überprüft und verarbeitet. Während dieser Zeit sind keine weiteren Benutzereingaben möglich.
- Nach dem Verarbeiten werden die Primfaktoren über RS232 an den Aufrufer übergeben. Wenn diese Übertragung abgeschlossen ist, liest das Programm wieder neue Daten ein.

Die tatsächliche Umsetzung entspricht nicht ganz dieser Beschreibung. Das nachfolgende Diagramm entspricht der tatsächlichen Implementierung:

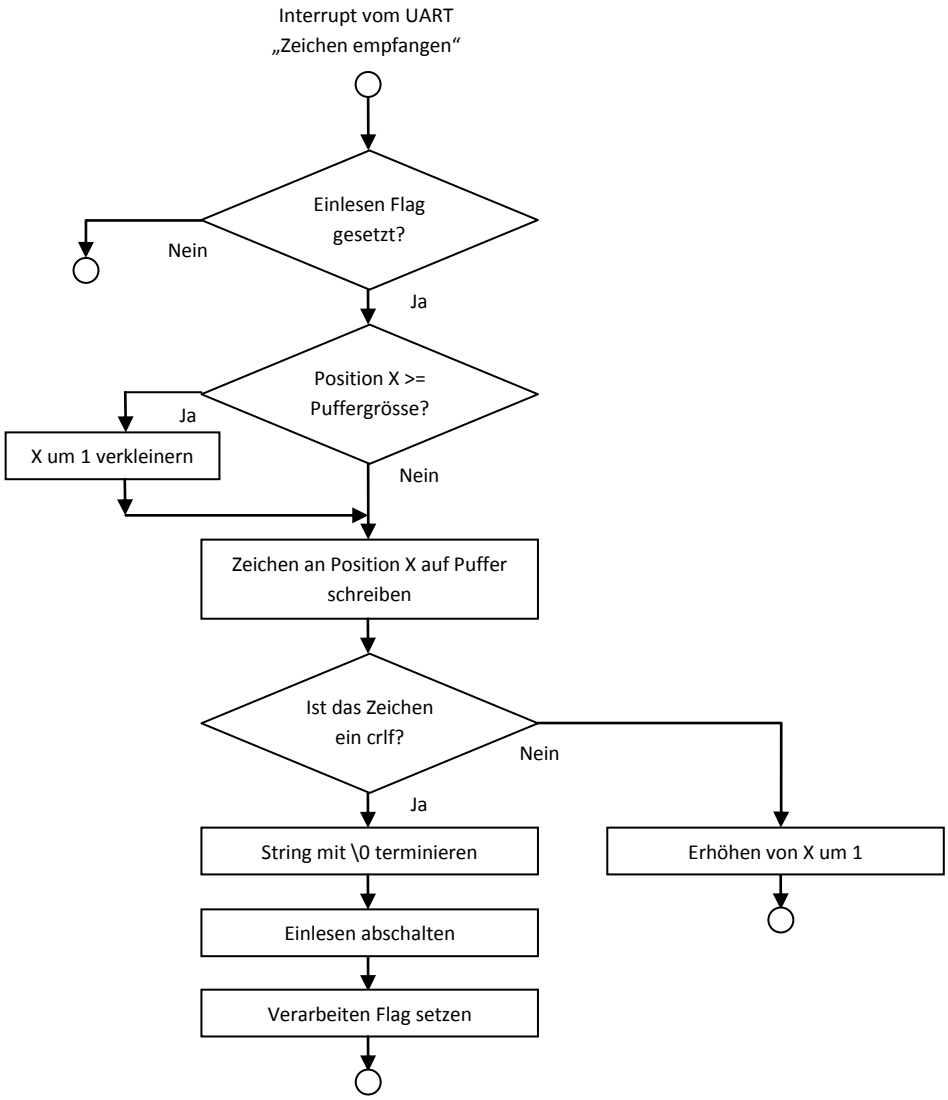


*: „crlf“ besteht eigentlich aus zwei Zeichen. Diesem Umstand wird in der Software Rechnung getragen. Dies hier darzustellen würde jedoch die Übersichtlichkeit stören.

UART reader

Dieses Modul ist in ASM implementiert. Es wird als Interrupt-Routine jedes Mal aufgerufen, wenn der UART ein neues Zeichen über RS232 empfangen hat.

Die *UART reader* Funktion ist wie folgt aufgebaut:



Solange das Einlesen-Flag gesetzt ist, wird jedes empfangene Zeichen dem Puffer hinzugefügt. Nach dem Hinzufügen wird überprüft, ob es sich um ein crlf handelt. Wenn das der Fall ist wird das weitere Einlesen verhindert und das Verarbeiten-Flag gesetzt.

Immer dann, wenn das empfangene Zeichen kein crlf ist, werden weitere zusammengehörige Zeichen folgen. Daher wird der Zeiger auf den Speicher um eine Adresse inkrementiert.

UART writer

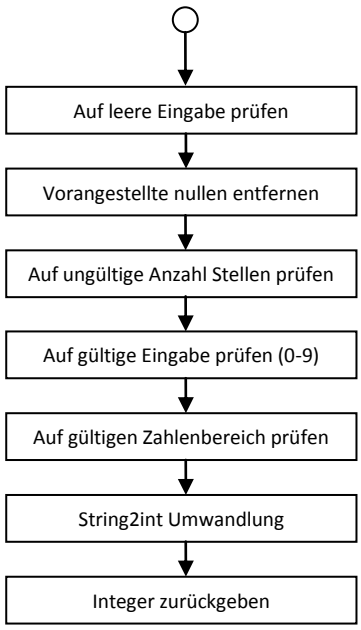
Dieses Modul ist in ASM implementiert. Es gibt einen String, welcher beim Funktionsaufruf als String übergeben werden kann, über die serielle Schnittstelle aus. Es wird dabei Zeichen für Zeichen übertragen, wobei jeweils zwischen zwei Zeichen gewartet wird, bis die nächste Übertragung stattfinden kann.

String2int

Dieses Modul ist in C implementiert. Es wird jedes Mal aufgerufen, wenn eine Eingabe durch den Benutzer erfolgt ist. Es dient dazu, die Eingabe des Benutzers auf ihre Gültigkeit zu prüfen und zu parsen.

Es sollen dabei nur positive Ganzzahlen im Definitionsbereich der Funktion (wie er weiter vorne in diesem Dokument erklärt wurde) als gültig betrachtet werden. Alle anderen Eingaben sollen zu dem uint32 Wert 0xFFFFFFFF führen.

Die *string2int* Funktion ist wie folgt aufgebaut:



Intarray2string

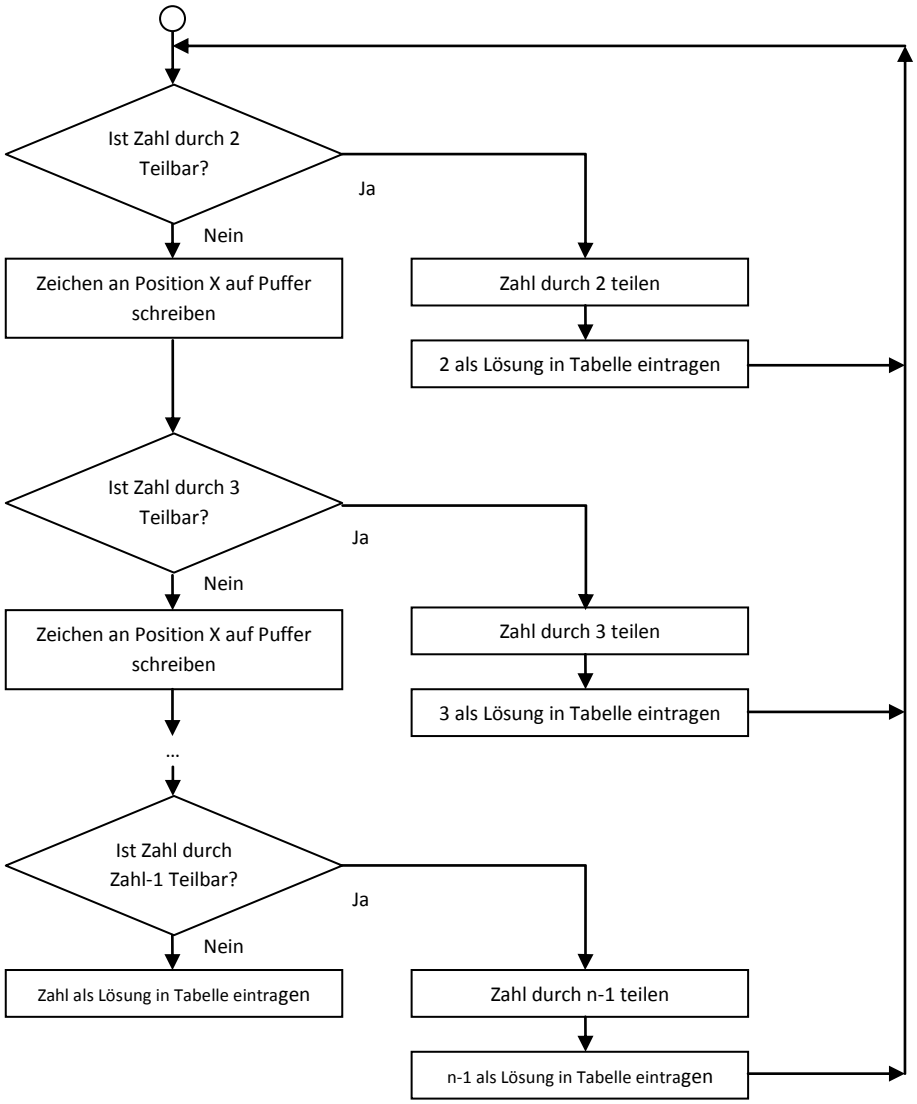
Dieses Modul ist in C implementiert. Es wird nach jeder Primfaktorzerlegung aufgerufen. Diese Funktion schreibt die gefundenen Primzahlen im Integer-Array in einen einzigen, mit Leerzeichen und *-Zeichen formatierten String.

Die I*ntarray2string* Funktion arbeitet mit sprintf() aus der stdio.h-Library und strcat() aus string.h.

factor

Dieses Modul ist in ASM implementiert. Es wird jedes Mal aufgerufen, wenn eine geparste und gültige Zahl in ihre Primfaktoren zerlegt werden soll.

Die *factor* Funktion ist wie folgt aufgebaut:



Alle Umgesetzten Lösungen basieren auf dem hier dargestellten Prinzip. Der Einzige Unterschied liegt in der Art der Divisoren. Beispielsweise kann durch jede Zahl kleiner als die zu zerlegende Zahl dividiert werden. Dies ist aber relativ wenig effizient. Es könnte auch nur durch die Ungeraden Zahlen und durch 2 dividiert werden. Ebenfalls könnte zuerst durch die grösste Zahl, die noch kleiner ist als die zu zerlegende Zahl, geteilt werden.

In Unserer Endversion der Software werden Ungerade Zahlen und die Zahl 2 als Divisoren angenommen, die kleiner als die aufgerundete Wurzel der zu zerlegenden Zahl sind. Denn es existiert keine Zahl, die einen Primfaktor enthält, der grösser ist als ihre eigene Wurzel. Beim Aufrunden wird in unserer Lösung aus Effizienzgründen nicht die nächst grössere Zahl, sondern die nächst oder übernächst grössere Zahl, die in der Reihe 2^n enthalten ist, verwendet.

div, mod, sqrt

Diese Funktionen wurden von uns in ASM implementiert. Hier sollen die Gedankengänge hinter deren Entwicklung kurz erläutert werden.

div, mod

Für die Division und die Modulo-Operation greifen wir auf einen Standard 32-bit Integer Division Algorithmus zurück. Siehe hierzu auch wiki.

sqrt

Für die Berechnung der Wurzel haben wir einen Algorithmus entwickelt, welcher sich der Logarithmen-Gesetze bedient. Er funktioniert etwa so:

$$\sqrt{317} = ?$$

$$317 = 2^{\frac{log(317)}{log(2)}} = 2^{8.3083}$$

$$\sqrt{317} = 317^{1/2} = 2^{\frac{1}{2}*\frac{log(317)}{log(2)}} = 2^{\frac{1}{2}*8.3083} = 2^{4.1542}$$

$$2^{4.1542} = \sqrt{317} = 17.8049$$

Dabei wird $2^{\frac{log(317)}{log(2)}}$ nicht exakt berechnet, sondern es wird die Anzahl signifikanter Stellen gezählt. Dies entspricht dem Aufgerundeten Wert von $\frac{log(317)}{log(2)}$, also in diesem Fall 9. Eine Division durch 2 mit anschliessender Addition von 1 liefert 5 als Exponent. Und $2^5 = 32$, also der nächst grössere Wert (als die exakte Wurzel), der in der Reihe 2^n enthalten ist.

Test

Im nachfolgenden Abschnitt werden die Tests beschrieben, welche an der Software im Endzustand durchgeführt wurden. Die Tests wurden vor dem implementieren der Software entwickelt.

Test	Soll	Ist	i.O.
Kann Hyperterminal-Sitzung mit Carme-Kit eröffnet werden?	Ja	Ja	i.O.
Wie reagiert die Software auf ein Enter ohne Eingabe? (Eingabe „“)	Fehlermeldung	Fehlermeldung	i.O.
Wie zerlegt die Software die Eingabe „0“?	„0“	„0“	i.O.
Wie zerlegt die Software die Eingabe „1“?	„1“	„1“	i.O.
Wie zerlegt die Software die Eingabe „4“?	„2 * 2“	„2 * 2“	i.O.
Wie zerlegt die Software die Eingabe „8“?	„2 * 2 * 2“	„2 * 2 * 2“	i.O.
Wie zerlegt die Software die Eingabe „741“?	„3 * 13 * 19“	„3 * 13 * 19“	i.O.
Wie zerlegt die Software die Eingabe „000017“?	„17“	„17“	i.O.
Wie zerlegt die Software die Eingabe „4294967294“? (2^32-2)	„2 * 2147483647“	„2 * 2147483647“	i.O.
Wie zerlegt die Software die Eingabe „-0“?	Fehlermeldung	Fehlermeldung	i.O.
Wie zerlegt die Software die Eingabe „-9“?	Fehlermeldung	Fehlermeldung	i.O.
Wie zerlegt die Software die Eingabe „12.0“?	Fehlermeldung	Fehlermeldung	i.O.
Wie zerlegt die Software die Eingabe „12.3“?	Fehlermeldung	Fehlermeldung	i.O.
Wie zerlegt die Software die Eingabe „a“?	Fehlermeldung	Fehlermeldung	i.O.
Wie zerlegt die Software die Eingabe „a12“?	Fehlermeldung	Fehlermeldung	i.O.
Wie zerlegt die Software die Eingabe „12a“?	Fehlermeldung	Fehlermeldung	i.O.
Wie zerlegt die Software die Eingabe „1a2“?	Fehlermeldung	Fehlermeldung	i.O.
Wie zerlegt die Software die Eingabe „4294967295“? (2^32-1)	Fehlermeldung	Fehlermeldung	i.O.
Wie zerlegt die Software die Eingabe „4294967296“? (2^32)	Fehlermeldung	Fehlermeldung	i.O.
Wie zerlegt die Software die Eingabe „4294967297“? (2^32+1)	Fehlermeldung	Fehlermeldung	i.O.
Wie reagiert die Software auf Eingaben mit mehr als 50 Zeichen?	Erste 50 Zeichen werden ausgewertet	Erste 50 Zeichen werden ausgewertet	i.O.
Wie reagiert die Software auf Eingaben mit genau 49 Zeichen? (z.B. „0000000000000000000000000000000000000000000000007“)	Wird ausgewertet („7“)	Wird ausgewertet („7“)	i.O.
Wie reagiert die Software auf Eingaben mit genau 50 Zeichen? (z.B. „00000000000000000000000000000000000000000000000007“)	Wird ausgewertet („7“)	Wird ausgewertet („7“)	i.O.

Die oben genannten Tests wurden am 19.01.2013 um 20:00 von N. Käser durchgeführt. Sämtliche Testkriterien wurden dabei beanstandungslos erfüllt.

Timing

Während der Entwicklung dieses Projekts wurden diverse Algorithmen getestet. Nachfolgend eine kurze Übersicht über die Ausführgeschwindigkeiten:

Zustand	Taktfrequenz des Rechners	Rechenzeit (ca.) [s]
1. Ansatz (Look-up-table, 0..2^16-1)	2.4GHz	0
2. Ansatz (entspricht aktuellem Algorithmus, nicht optimiert, 0..2^16-1)	2.4GHz	0.1
2. Ansatz (zusätzlich 0..2^32-1)	2.4GHz	25
2. Ansatz (erste Optimierung -> Verzicht auf gerade Divisoren ausser 2)	2.4GHz	10
3. Ansatz (Einschränken des Suchraumes mittels Wurzel-Operation)	**3.4GHz**	0.1
3. Ansatz (übersetzen in ASM, ausführen mittels Simulation)	2.4GHz	2.5
3. Ansatz (Ausführung auf CARME-Kit)	**500MHz**	0.3

Fazit

Wir haben im Verlauf dieser Projektarbeit die Möglichkeit erhalten, diverse Implementierungen eines Algorithmus zu testen. Unser Hauptaugenmerk lag dabei auf der Ausführungsgeschwindigkeit. Es kristallisierten sich aber auch kleine Teilprobleme heraus, deren Schwierigkeit wir anfänglich unterschätzt hatten. Namentlich das Arbeiten mit der Hochsprache C mit anschliessendem Übersetzen zu ASM bringt viele Vorteile, jedoch auch einige Stolpersteine mit sich. So sei hier beispielhaft die Berechnung einer Wurzel als unterschätztes Teilproblem erwähnt; in C steht dazu eine Library-Funktion zur Verfügung. In ASM hingegen muss diese Funktion von Hand implementiert werden. Dieser Umstand bietet aber gleichzeitig die Möglichkeit, gewisse Überlegungen zu überdenken. So haben wir uns schlussendlich für eine Lösung entschieden, die die Wurzel nur annähert, dafür aber ohne Fliesskommazahlen auskommt (und alles in allem nur ca. 10 Taktzyklen in Anspruch nimmt). Natürlich wird der Algorithmus durch diese Näherung etwas langsamer, doch die Ersparnis beim Umgang mit der exakten Lösung der Wurzel kompensiert diesen Mehraufwand.

Alles in allem war dieses Projekt für alle beteiligten sehr lehrreich. Es hat uns auch daran erinnert, dass sich hinter jeder noch so einfach anmutenden Operation - welche man in einer Hochsprache als selbstverständlich voraussetzt - eine nicht zu unterschätzende Portion Denkarbeit versteckt.

Mögliche Weiterführung

In zukünftigen Versionen könnte mit vertretbarem Aufwand folgendes realisiert werden:

- Definitionsbereich auf 0…2^{32}-1 steigern (also 2^{32}-1 auch zulassen)
- Statusanzeige an den LEDs des Carme-Kits
- Zu zerlegende Zahl und/oder Ergebnis auf dem LCD des Carme-Kit darstellen
- Die in C implementierten Softwareteile könnten nach ASM übersetzt werden.

Mit etwas grösserem Aufwand wäre beispielsweise folgendes zu realisieren:

- Definitionsbereich auf 0…2^{64}-1 steigern
- Ungenutzte Rechenzeit verwenden, um eine Look-Up-Table zur Laufzeit zu berechnen. Dadurch werden spätere Anfragen unter Umständen erheblich beschleunigt (Hauptsächlich sinnvoll in Kombination mit grösserem Definitionsbereich)
- Eingabe-Puffer implementieren, so dass während der Berechnung bereits eine weitere Anfrage gesendet werden kann (Ebenfalls nur in Kombination mit grösserem Definitionsbereich wirklich sinnvoll)

An der bestehenden Version kann bei Bedarf unter anderem folgendes geändert werden:

- Der eigentliche Algorithmus kann zu Testzwecken ausgetauscht werden