Bericht

KU Rechnerorganisation

2018

Lukas Hirschhuber ([l.hirschhuber@student.tugraz.at](mailto:l.hirschhuber@student.tugraz.at)) **11705228**

Zdenek Zeman([zzeman@student.tugraz.at](mailto:zzeman@student.tugraz.at)) **0071074**

# 1.Einleitung

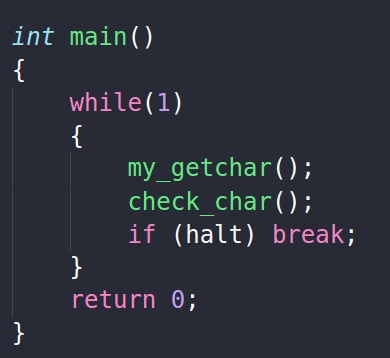
Dieser Bericht stellt eine Zusammenfassung der Arbeiten an Aufgabe 4 für Rechnerorganisation im Sommersemester 2018 dar. Die Aufgabe befasste sich mit der Erstellung eines Programmes für Toy Deluxe, welche wie in Aufgabe 1 mit der RPN funktioniert. Das Programm sollte um Multiplizieren erweitert werden….

Der Input für Toy Deluxe sollte mit einem Keyboard realisiert werden. Der Output sollte über Binary to ASCII (B2A) auf einem Terminal dargestellt werden. B2A aus Aufgabe 3 sollte dahingehend angepasst werden, dass die Zeichen „+“, „\*“ und „\n“ dargestellt werden können.

Als Grundlage für das die Erstellung des Programms wurde das Programm von Zdenek Zeman aus Aufgabe 1 benutzt. Das zur Verfügung gestellte Toy Deluxe Modell wurde an die Aufgabe angepasst.

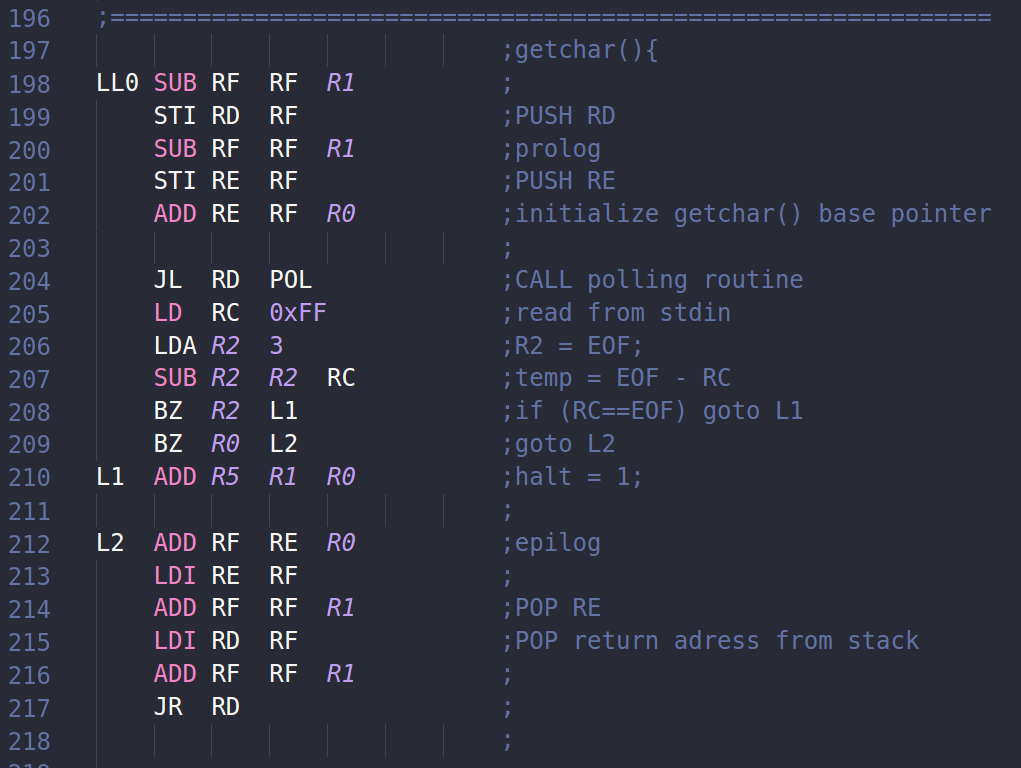
# 2.Erstellung des Programms

Als erster Schritt wurde erstmal ein der Spezifikation entsprechendes C-Programm geschrieben. Jedoch war der Programmierer etwas übermotiviert und wollte nicht nur die Funktionen „getchar()“ und „printf()“ implementieren, sondern auch noch weitere Funktionen wie z.B. „add()“ oder „save()“. Die „main()“ beinhaltete also nur die Funktionsaufrufe „getchar()“ und „check\_char()“ gefolgt von einer Abbruchbedingung. Dieser Umstand sollte sich noch als problematisch herausstellen.

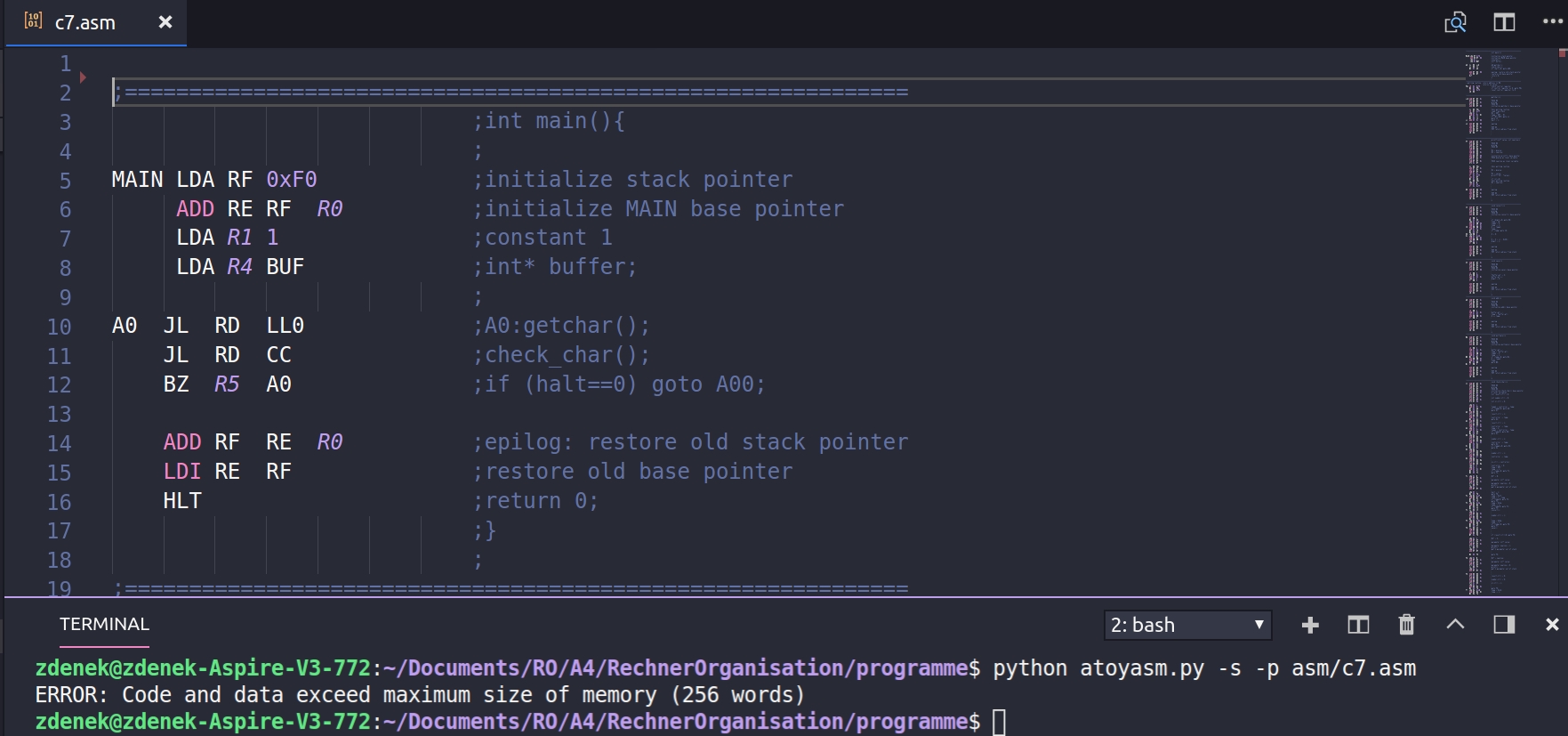


Weiters wurde der Code zum Kompilieren vorbereitet. Schleifen und if-statements wurden in goto statements umgewandelt und Registernamen für die Variablen verwendet.

Eine große Hürde war dann die Implementierung des Stacks und der entsprechenden Pointer. Nach intensivem Studium der LV-Unterlagen und der Entscheidung, dass die Funktions-Return-Adresse vor dem Prolog am stack gepusht bzw. nach dem Epilog vom Stack gepopt wird, wurde diese Hürde auch genommen.



Die Herangehensweise war so, dass der C-Code Schritt für Schritt kompiliert und getestet wurde. Als jedoch alle Funktionen fertig waren und zusammengefügt wurden, kam ein Problem auf.



So schnell stoßt man an die Grenze eines 1 kB Speichers. Es war aber klar, dass es daran lag, dass das Programm neben der „main()“ noch 7 zusätzliche Funktionen beinhaltete. Die entsprechenden Prologe und Epiloge verbrauchten einfach zu viel Speicher. Es wurde also auf drei Funktionen („main()“, „getchar()“ und „printf()“) reduziert.

Getestet wurde mithilfe eines shell\_scripts mit folgenden Instruktionen: .toy erstellen, mit Test-Input simulieren, die Simulation in einer .txt Datei ausgeben und die letzten beiden Zeilen der Simulation (Toy-STDOUT) mit dem richtigen Output vergleichen. Die 2. bis letzte Anweisungen wurden mit jedem Test-Fall ausgeführt. Dies erleichterte die Arbeit enorm, da man nun jede Änderung im ASM-Code mit einem einfachen Terminal-Befehl schnell und umfangreich testen konnte.

Im Zuge der finalen Testphase wurde ein interessantes Problem entdeckt. Wie weiter oben zu sehen, wurde der Stack-Pointer mit 0xF0 initialisiert. Dies musste jedoch geändert werden, da der Stack-Pointer auf Speicherplätze des Buffers zugriff. Als Initialisierungsadresse wurde also etwas weiter hinten im Speicher 0xF4 gewählt. Hier befindet sich auch die Schwachstelle des Programms, die bisher noch nicht behoben werden konnte.

# 3.Anpassung von Toy Deluxe

Als Grundlage wurde das zur Verfügung gestellte Logisim Programm von Toy Deluxe verwendet ( Abbildung 1)

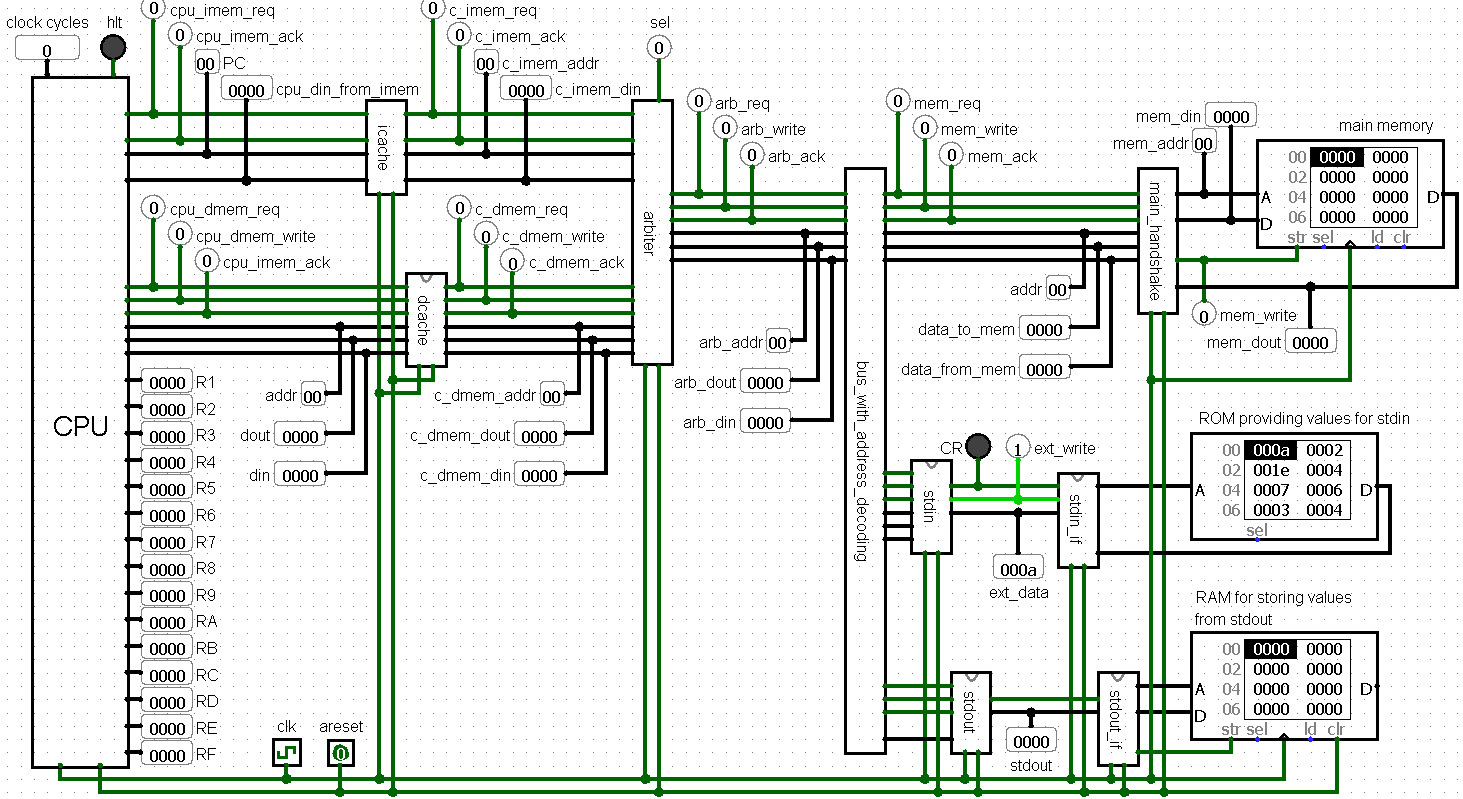


Abbildung 1

Der ROM und der RAM wurden zusammen mit ihren dazugehörigen Interfaces aus dem Modell entfernt. (Abbildung 2)

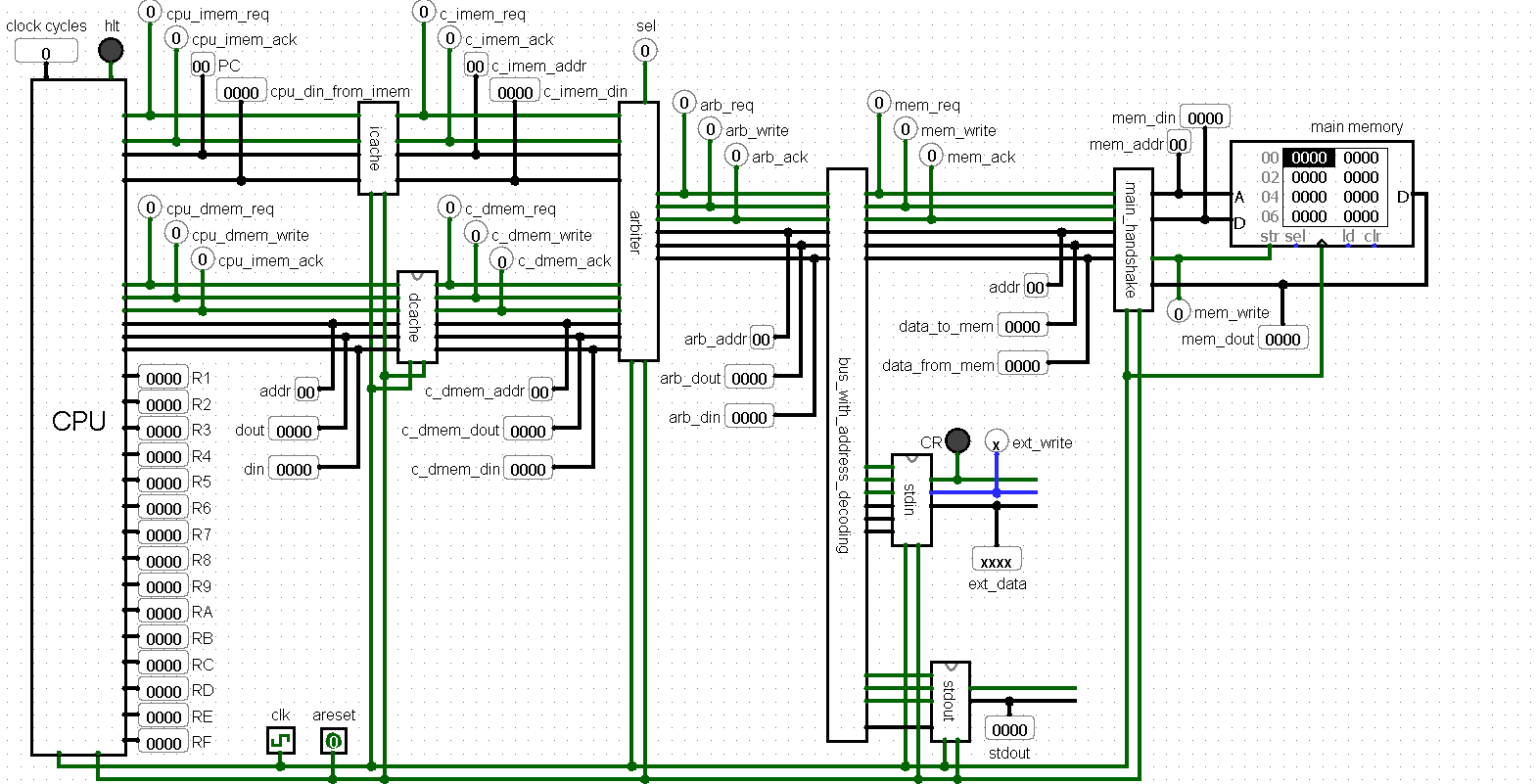


Abbildung 2

Zuerst wurde das Keyboard mit dem dazugehörigen Interface der Schaltung hinzugefügt

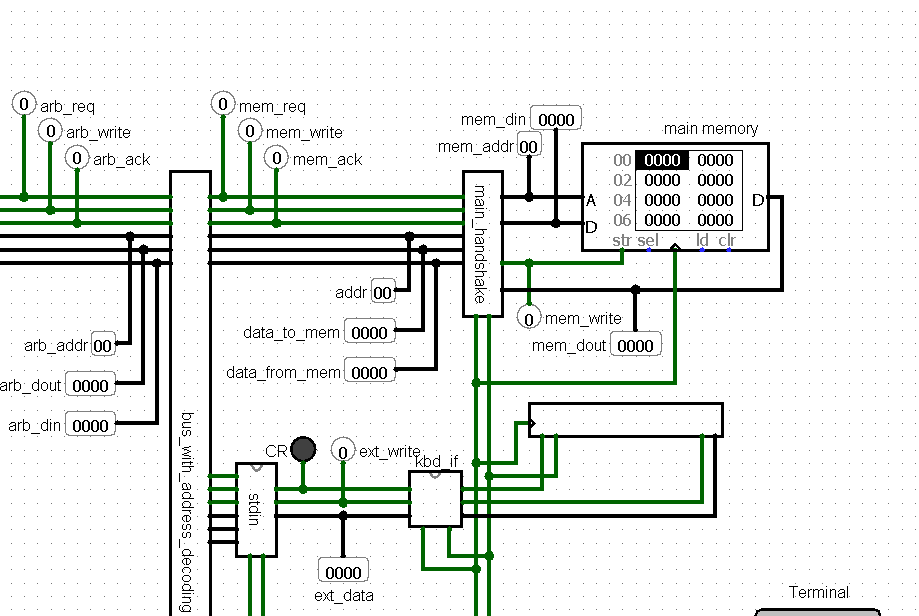


Abbildung 3 Mit angeschlossenem Keyboard

Im Keyboardinterface wird das 7-Bit Datensignal mit führenden Nullen auf ein 16-Bit erweitert, da unser Toy mit 16-Bit rechnet. Ebenfalls im Keyboardinterface wird das Signal, dass etwas im Puffer des Keyboards vorhanden ist weitergegeben. Über den Ausgang Read\_ok wird das Herausnehmen aus dem Puffer verhindert.

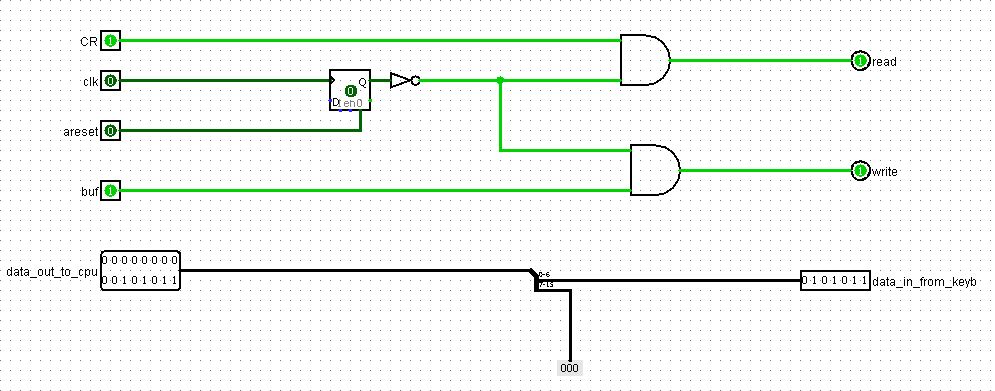


Abbildung 4 Keyboard-Interface

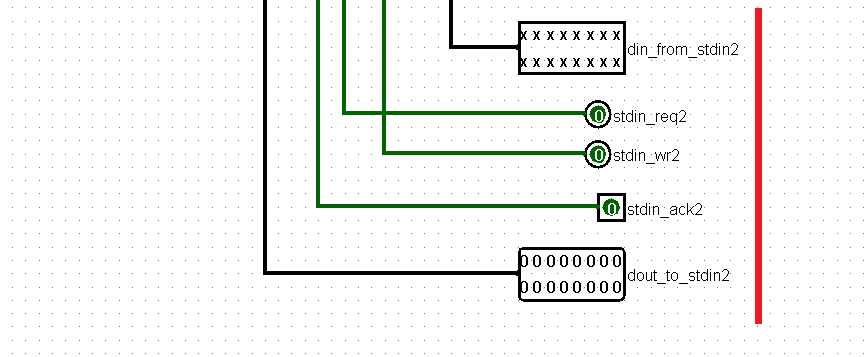
Als nächstes musste das Modell um einen weiteren Stdin erweitert werden. Hier wurde der Bus erweitert. Dies Pins wurden mit den bereits vorhandenen im Bus verbunden.

Abbildung 5 Erweiterter Bus

Stdin2 wurde nach Grundlage vom gegebenen Stdin angepasst.

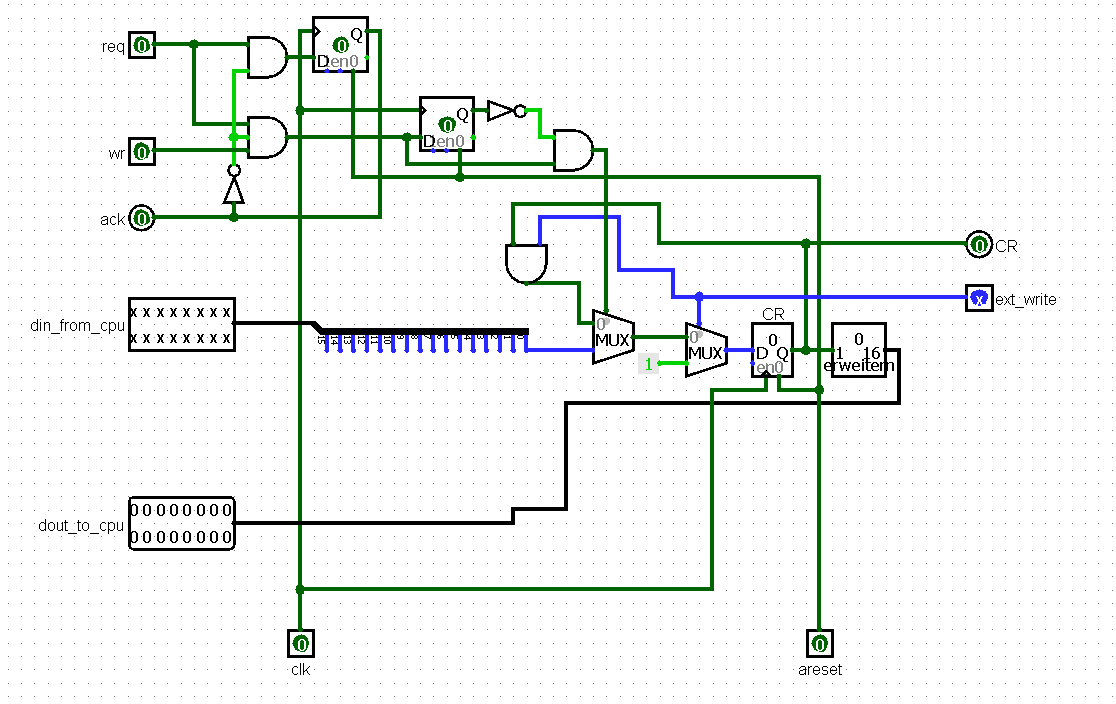


Abbildung 6 Stdin2

Stdin2 wird mit dem Bus verbunden

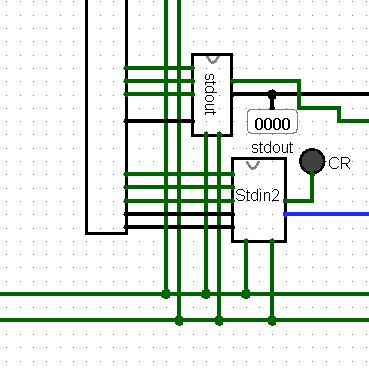
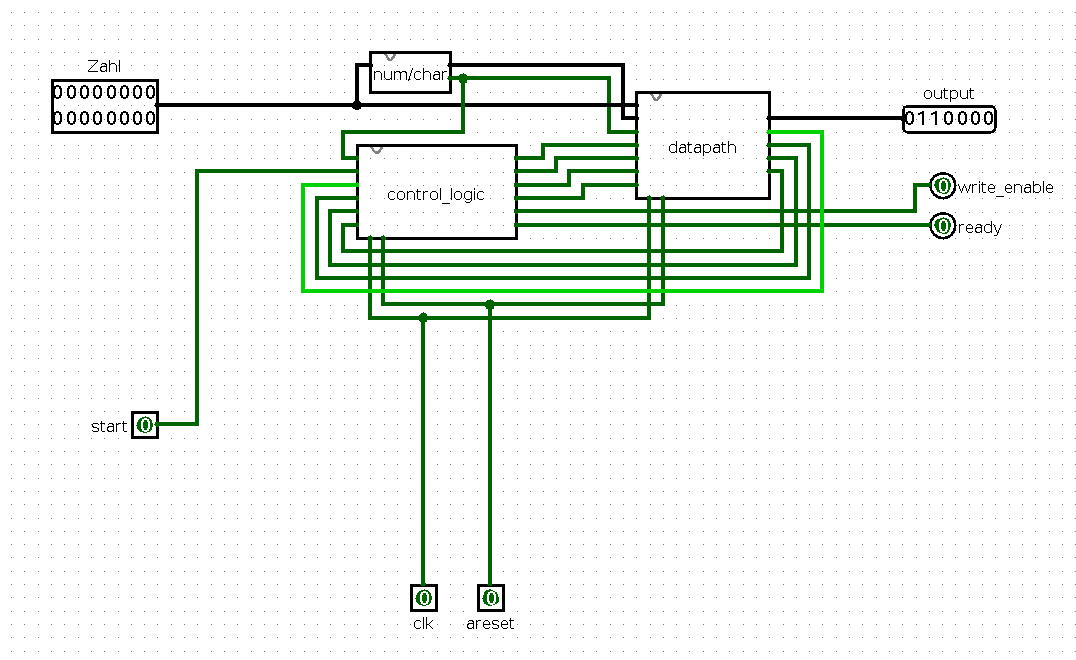
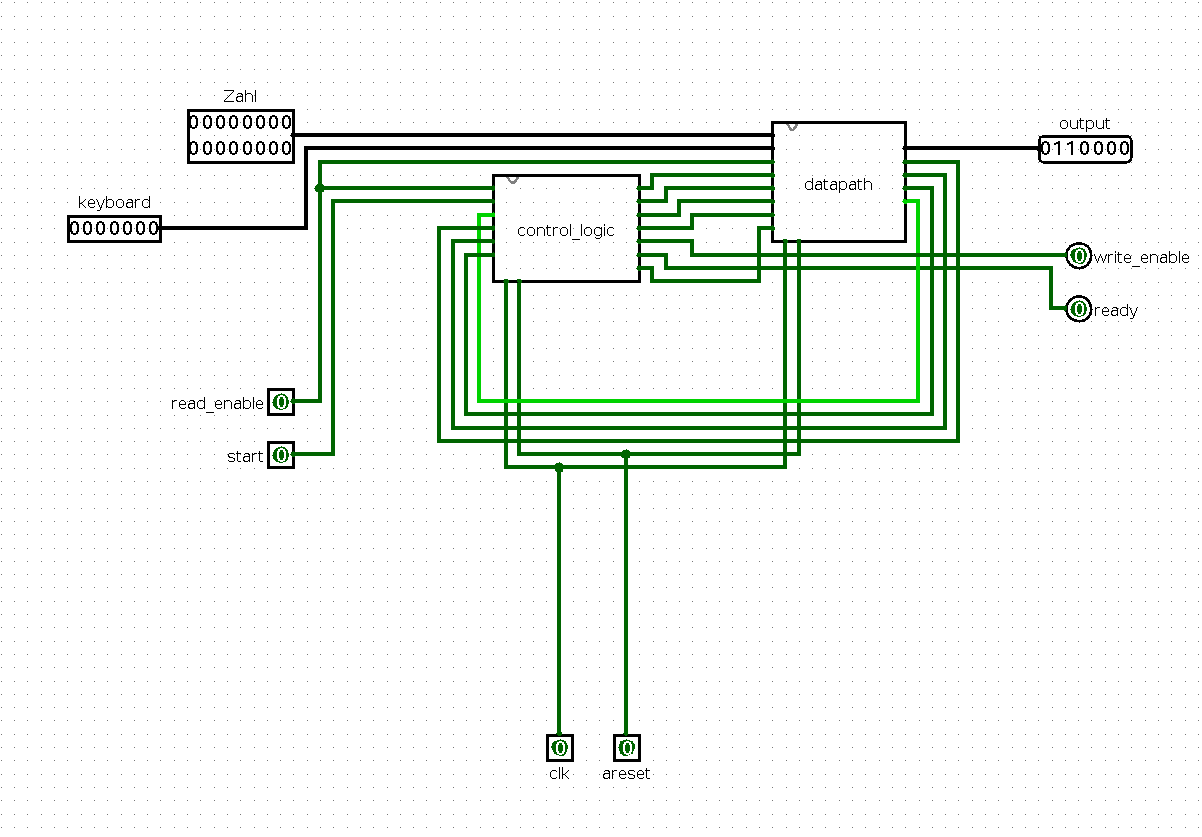


Abbildung 7 Stdin2 an Bus

# 4.Binary2Ascii erweitern

Es wurde das Binary2Ascii aus Aufgabe 3 modifiziert, um der Spezifikation zu entsprechen.



Bei read\_enable = 1 wurde der Tastatur-Input unverändert ausgegeben. Bekommt num/char eine führende 1 im Input, wird ein Signal an datapath und control\_logic gesendet. Es wird also das read\_enable-Signal der Vorgängerversion emulgiert, was bewirkt, dass datapath die Daten aus num/char kommend ausgibt. Num/char schaltet der Kodierung entsprechend auf das richtige Zeichen („\n“, „+“ oder „\*“).

