UNIVERSITÄT
LEIPZIG

# Der Minirechner 2i

Handbuch und
Bedienungsanleitung
Version 1.21

Max Braungardt
Thomas Schmid

Abteilung Neuromorphe
Informationsverarbeitung
Institut für Informatik

**Kontakt**

Max Braungardt
Augustusplatz 10 | Raum P522
04109 Leipzig

E-Mail: braungardt@informatik.uni-leipzig.de

Thomas Schmid
Augustusplatz 10 | Raum P533
04109 Leipzig

E-Mail: schmid@informatik.uni-leipzig.de



Der Minirechner 2i wurde an der Universität Tübingen von Werner Dreher entwickelt und ist seit ca. 2010 Bestandteil des Praktikums im Modul „Grundlagen der Technischen Informatik 2“ der Universität Leipzig. Diese Bedienungsanleitung wurde von der Abteilung Neuromorphe Informationsverarbeitung der Universität Leipzig erstellt.

# Inhaltsverzeichnis

# 1 Überblick

Der Minirechner 2i ist ein einfacher, aber vollständiger 8-Bit-Rechner. Er realisiert eine von-Neumann-Architektur, weshalb sich im grundsätzlichen Aufbau insbesondere Datenpfad und Steuerwerk unterscheiden lassen. Funktionen und Berechnung lassen sich auf dem Minirechner 2i durch binär kodierte Befehlswörter realisieren.

Physikalisch besteht die Minirechner-Plattform aus drei Platinen:

1. einer Logikplatine, die im Wesentlichen ein Field Programmable Gate Array (FPGA) und ein Random Access Memory (RAM) für Daten enthält;
2. einer Display-Platine, auf der sich ca. 300 LEDs befinden;
3. einer Erweiterungsplatine mit 2 Digital-Analog-Wandlern, welche durch den Minirechner 2i jedoch nicht angesprochen werden können.

Die CPU mit Datenpfad und Steuerwerk ist im FPGA realisiert. Der Daten-RAM ist über den Memory-Bus an die CPU angekoppelt. Dieser Bus besteht aus folgenden Leitungen:

- 8 Datenleitungen von der CPU zum Daten-RAM (Memory Data Out: MEMDO)
- 8 Datenleitungen vom Daten-RAM zur CPU (Memory Data In: MEMDI)
- 8 Adressleitungen (Memory Address: MEMA)
- 3 Steuerleitungen (Chip Enable: CE, Output Enable: OE, Write Enable: WE)

Außer dem Daten-RAM hängen an diesem Bus noch vier Input-Register („in FC“ bis „in FF“ auf Adresse FC bis FF), 2 Output-Register („out FE“ und „out FF“ auf Adresse FE bzw. FF) und eine serielle Schnittstelle (Universal Asynchronous Receiver and Transmitter: UART, Adressen FA und FB). Die Input-Register können mit den Tastern bitweise beschrieben werden und dienen zur Eingabe von Daten an die CPU; sie können von der CPU nur gelesen werden. Die Output-Register können durch die CPU beschrieben werden und dienen zur Visualisierung von Ergebnissen durch LEDs. Über den UART kann die CPU z.B. mit einem PC kommunizieren. Die Input- und Output-Register und der UART befinden sich innerhalb des FPGAs, der Daten-RAM ist ein eigener Baustein. Das Daten-RAM, die darin enthaltenen Input-Register sowie die Output-Register sind im Blockschaltbild (Abbildung 1.1) nicht enthalten, da sie nicht Teil der eigentlichen CPU sind, sondern als zusätzliche Bauelemente zur Verfügung stehen.

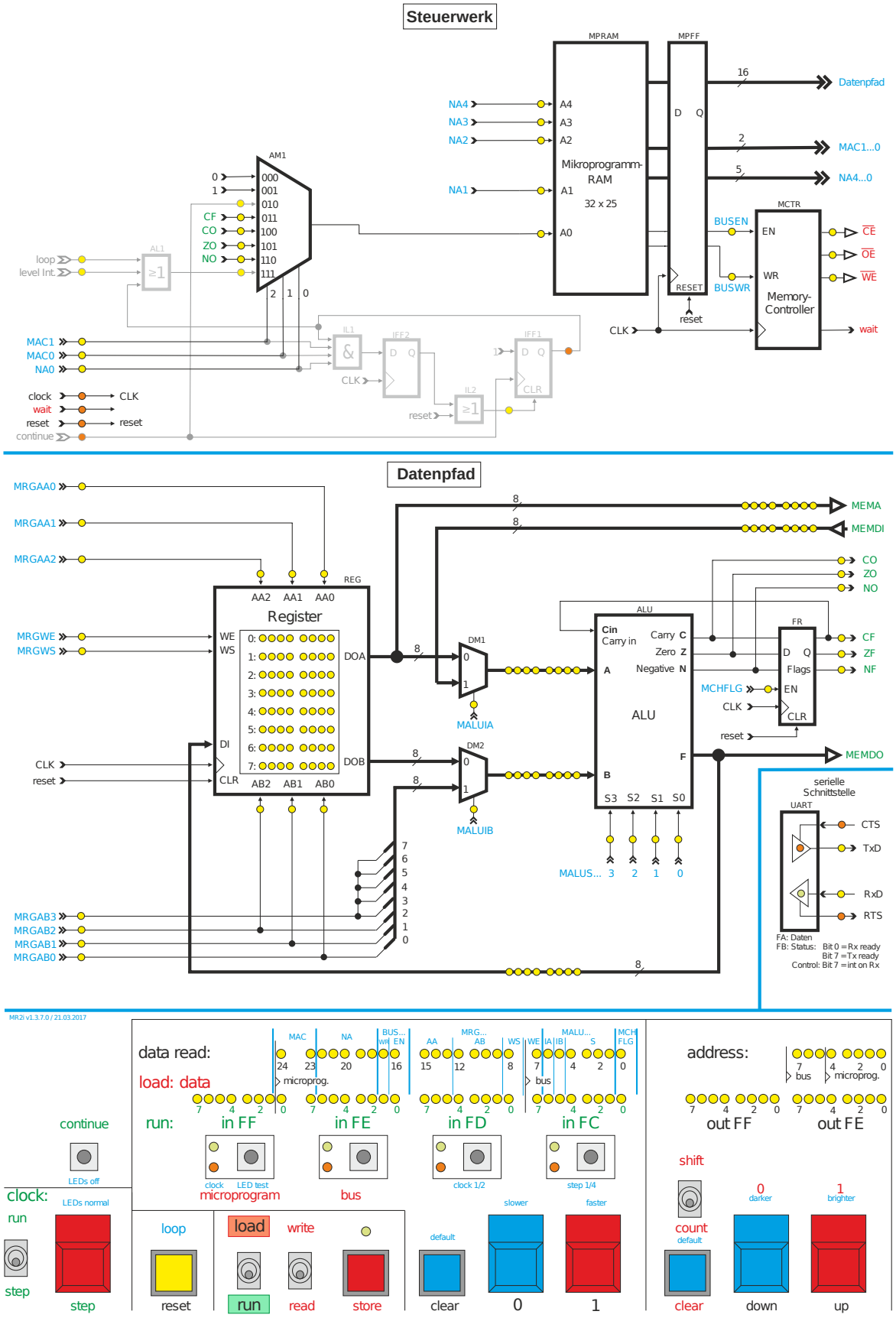


Abbildung 1.1: Blockschaltbild des Minirechner 2i.

# 2 Datenpfad

Der Datenpfad kann 8-Bit-breite Daten verarbeiten und besteht aus einem Register-Block und einer arithmetisch-logischen Einheit (Arithmetic Logic Unit: ALU). Der Register-Block beinhaltet acht universelle Register zu je acht Bit, in denen sowohl Daten (z.B. Zwischenergebnisse), als auch Daten-RAM-Adressen gespeichert werden können. Die ALU stellt 16 Funktionen zur arithmetischen und logischen Verknüpfung der Eingänge A und B zur Verfügung. Vor diesen beiden Eingängen befindet sich jeweils ein 2-zu-1-Multiplexer (Abbildung 2.1), mit denen die für die ALU bestimmten Daten aus je zwei Quellen ausgewählt werden können. Als Quellen kommen dabei der Register-Block, der Daten-RAM, sowie das Mikroprogramm-Wort (Eingabe von Konstanten) in Frage. Die Abbildung 2.1 zeigt eine Übersicht über den Datenpfad.

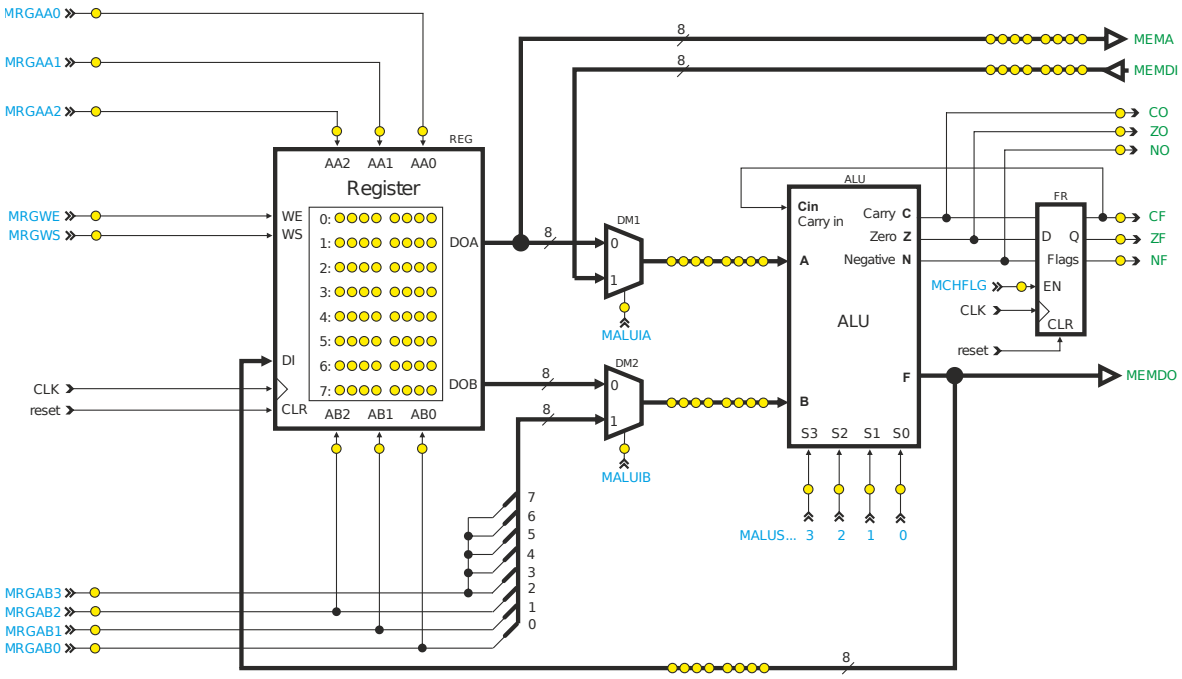


Abbildung 2.1: Schematische Darstellung des Datenpfads.

## 2.1 Registerblock

Der Register-Block (Abb. 2.2) kann 8-Bit-breite Daten, wie (Zwischen-)Ergebnisse und Adressen speichern. Er besitzt einen 8-Bit-breiten Daten-Eingang (Data In: DI) und die zwei 8-Bit-breite Daten-Ausgänge Data Out A (DOA) und Data Out B (DOB). Mit Hilfe der zwei 3-Bit-breiten Adresseingänge Address A (AA0, AA1, AA2) und Address B (AB0, AB1, AB2) kann ausgewählt werden, welche Register an die Ausgänge DOA und DOB angelegt werden sollen. Sollen Daten in eines der Register geschrieben werden, so muss mit Write Select (WS) ausgewählt werden, welche der beiden Adressen (AA oder AB) für die Auswahl des zu beschreibenden Registers verwendet werden soll. Wird der Eingang Write Enable (WE) aktiviert, so werden die am 8-Bit-breiten Eingang DI anliegenden Daten bei der nächsten aktiven Taktflanke in das ausgewählte Register geschrieben.

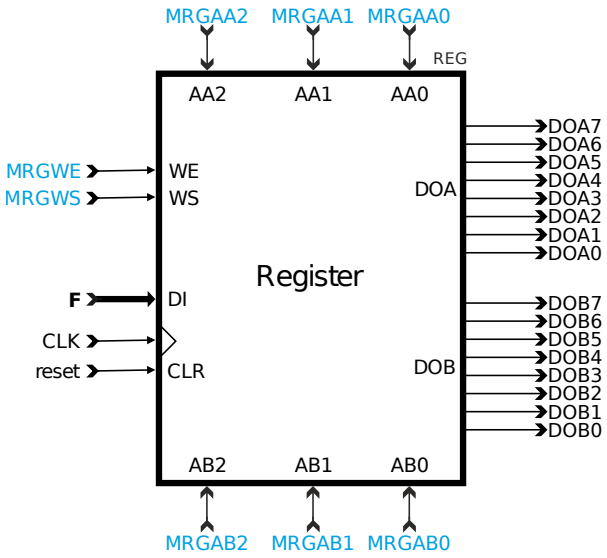


Abbildung 2.2: Schematische Darstellung des Registerblocks.

## 2.2 Arithmetisch-logische Einheit (ALU)

Die ALU ist für die arithmetische bzw. logische Verknüpfung zweier Zahlen zuständig. Als Datenquelle für den Eingang A der ALU kann entweder der Ausgang DOA des Registerblocks oder der Daten-RAM-Bus (Memory Data In: MEMDI) dienen (Auswahl durch den entsprechenden Steuereingang (Microprogram-Bit ALU Input A select: MALUIA) des Multiplexers vor Eingang A). Als Datenquelle für den Eingang B der ALU kann der Ausgang DOB des Registerblocks oder eine Konstante zwischen -8 und 7 (8-Bit-Zweierkomplement) dienen (Auswahl durch MALUIB, die Konstante wird durch die Steuereinheit bestimmt). Die Adressen für den Daten-RAM (Adressbus) kommen immer aus dem Ausgang DOA.

Die ALU enthält drei arithmetische bzw. logische Funktionseinheiten (Abbildung 2.3), mit denen man binäre und unäre Operationen durchführen kann:

- das 8-Bit-breite NOR U1 (bestehend aus 8 NORs mit je 2 Eingängen)
- den 8-Bit-Volladierer U2 (mit Carry-Eingang Cin und Carry-Ausgang C)
- den Shifter U3, der den Wert um ein Bit nach rechts versetzt durchreicht.

Mit Hilfe des Multiplexers U4 wird der gewünschte Wert ausgewählt und an den Ausgang F gelegt. Der Shifter ist in Wirklichkeit keine eigene Schaltung, sondern der Eingang A um ein Bit versetzt an den Multiplexer U4 angeschlossen. Der Multiplexer U5 wählt das Carry-Bit

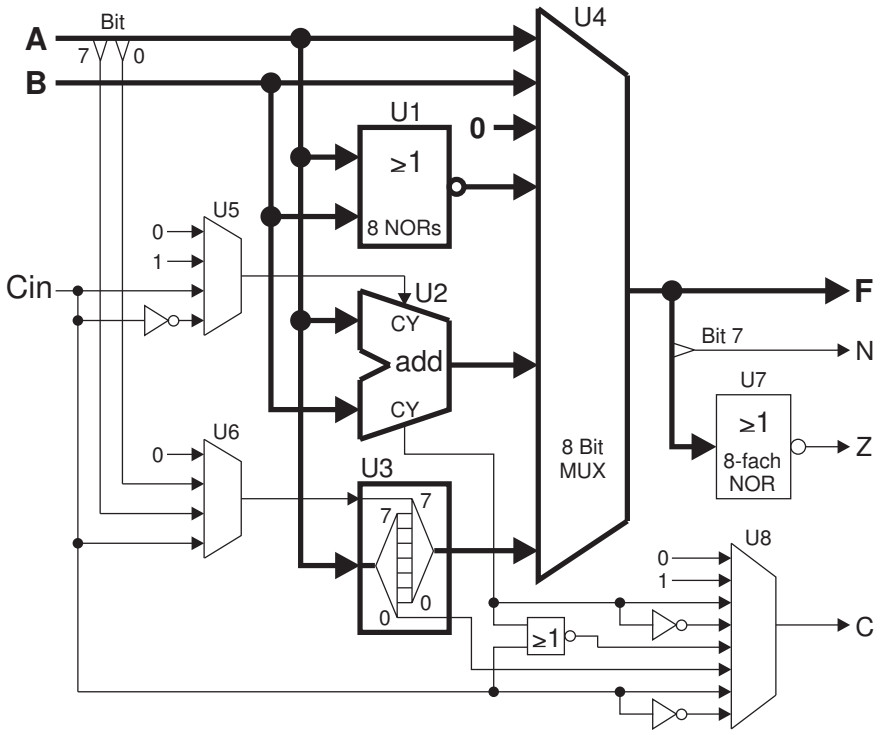


Abbildung 2.3: Blockschaltbild der ALU (dicke Leitungen enthalten acht Bit).

| Steuer-eingänge | Befehl allg. | Befehl bei $B = A$ | Funktion | C | N | Z | Bemerkung |
|---|---|---|---|---|---|---|---|
| 00 00 | ADDH | LSLH | $F = A + B$ | $OR$ | * | * | **add** and **h**old carry: $C = Cin \vee C$ |
| 00 01 | A | - | $F = A$ | 0 | * | * | Eingang **A** durchreichen |
| 00 10 | NOR | COM | $F = A$ NOR $B$ | 0 | * | * | bei $B = A$: **com**plement |
| 00 11 | 0 | - | $F = 0$ | 0 | 0 | 1 | Ergebnis immer 0 |
| 01 00 | ADD | LSL | $F = A + B$ | $Ca$ | * | * | bei $B = A$: **l**ogical **s**hift **l**eft |
| 01 01 | ADDS | (SL1) | $F = A + B + 1$ | $\overline{Ca}$ | * | * | **add** for **s**ubtraction<br>bei $B = A$: **s**hift **l**eft, rechts **1** einschieben |
| 01 10 | ADC | RLC | $F = A + B + Cin$ | $Ca$ | * | * | **ad**d with **c**arry<br>bei $B = A$: **r**otate **l**eft through **c**arry |
| 01 11 | ADCS | - | $F = A + B + \overline{Cin}$ | $\overline{Ca}$ | * | * | **ad**d with **c**arry for **s**ubtraction |
| 10 00 | LSR | - | $F(n) = A(n+1), F(7) = 0$ | $A(0)$ | * | * | **l**ogical **s**hift **r**ight, links 0 einschieben |
| 10 01 | RR | - | $F(n) = A(n+1), F(7) = A(0)$ | $A(0)$ | * | * | **r**otate **r**ight |
| 10 10 | RRC | - | $F(n) = A(n+1), F(7) =$Cin | $A(0)$ | * | * | **r**otate **r**ight through **c**arry |
| 10 11 | ASR | - | $F(n) = A(n+1), F(7) = A(7)$ | $A(0)$ | * | * | **a**rithmetic **s**hift **r**ight |
| 11 00 | B<br>CLC | - | $F = B$ | 0 | * | * | Eingang **B** durchreichen<br>**cl**ear **c**arry flag |
| 11 01 | SETC | - | $F = B$ | 1 | * | * | **set c**arry flag |
| 11 10 | BH | - | $F = B$ | $Cin$ | * | * | **B** and **h**old carry flag |
| 11 11 | INVC | - | $F = B$ | $\overline{Cin}$ | * | * | **inv**ert carry flag |

A, B = Dateneingänge, F = Ergebnis, C = carry out, N = negative out, Z = zero out, * = entsprechend dem Ergebnis F; Cin = Carry input in ALU, Ca = Carry aus Addierer, $\overline{x}$ = Signal $x$ invertiert

Tabelle 2.1: Funktionen der ALU.

für den Volladdierer, während die Einheit U6 entscheidet, ob und welches Bit beim Rechtsshift links eingeschoben wird. Das achtfach-NOR U7 analysiert den Ausgang F und liefert eine 1, falls dieser 0 ist. Der Multiplexer U8 bestimmt das Carry-Flag.

Die ALU ist ein reines Schaltnetz, weshalb der berechnete Wert innerhalb des gleichen Taktes am Ausgang F zur Verfügung steht. Die ALU besitzt außerdem drei Flag-Ausgänge:

- C = Carry = arithmetischer Überlauf oder rausgeschobenes Bit beim Shiften
- Z = Zero = Ergebnis ist 0
- N = Negative = Ergebnis ist eine negative Zahl (= Bit 7 ist gesetzt)

Diese Flag-Ausgänge können in dem 3-Bit-breiten Register „Flags“ gespeichert werden. Dieses Register übernimmt die Werte am Eingang D mit steigender Taktflanke, wenn gleichzeitig am Eingang Enable (EN) eine 1 anliegt. Wenn EN = 0 ist, bleibt der Ausgang Q unverändert. Das zwischengespeicherte Carry-Bit ist an den Carry-Eingang der ALU zurückgeführt und kann für Berechnungen mit mehr als acht Bit Datenwortbreite benutzt werden. Die Multiplexer in der ALU werden durch die Eingänge Select 0 bis 3 (S0 bis S3) gesteuert.

## 2.3 Ein- und Ausgabe

Programmierung und Programmausführung werden durch eine Reihe von Tastern gesteuert, die sich im Bedienfeld (Abbildung 2.4) befinden. Rote Beschriftungen gelten nur im Programmiermodus, grüne nur im Run-Modus, schwarze immer. Farbig beschriftete Schalter oder Taster haben im jeweils anderen Modus keine Funktion. Für eine ausführliche Beschreibung der Tasterfunktionen siehe Abschnitte 4.2 und 4.3.

Die 32 LEDs in der Zeile „load: data“ zeigen im Programmiermodus den Inhalt des Eingaberegisters an. Im Run-Modus wird diese Anzeige in die vier Input-Register FC, FD, FE und FF aufgeteilt.

Die 25 LEDs mit der Bezeichnung „data read“ zeigen im Programmiermodus (bei Auswahl „microprogram“) und im Run-Modus den Inhalt des Mikroprogramm-RAMs an der eingestellten Adresse. Ist im Programmiermodus „bus“ ausgewählt, so wird der Inhalt des Daten-RAMs des gewählten Input-Ports, oder das Daten- bzw. Statusregisters des UARTs an der eingestellten Adresse angezeigt.

Die 8 LEDs mit der Bezeichnung „address“ zeigen die eingestellte Adresse und die 16 LEDs „out FE“ und „out FF“ den Inhalt der Output-Register.

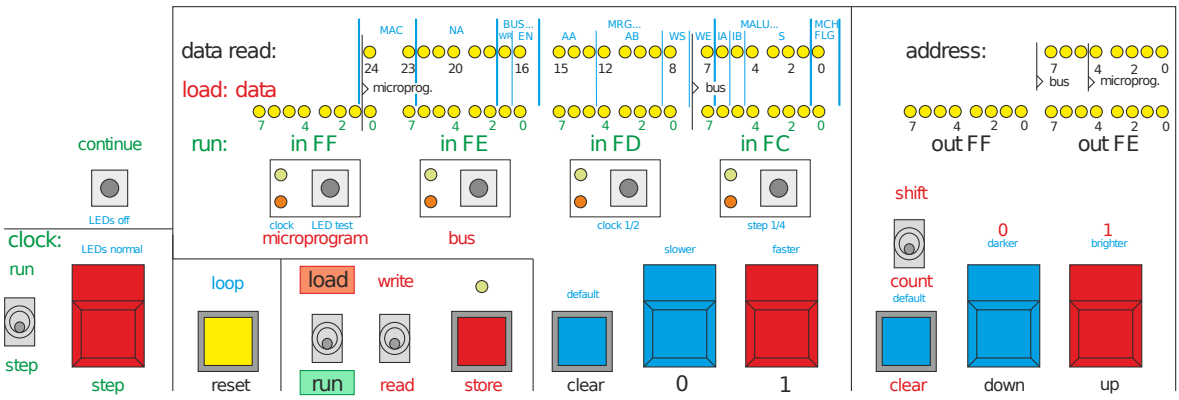


Abbildung 2.4: Schematische Darstellung des Bedienfelds.

# 3 Steuerwerk

Das Steuerwerk des Minirechner 2i arbeitet die Befehle aus dem Mikroprogramm-RAM ab und steuert andere Funktionseinheiten durch Steuersignale. Es setzt sich im Wesentlichen aus einer Adress-Dekodierung, dem Mikroprogramm-RAM und einem Memory-Controller zusammen. Darüber hinaus existiert ein Handling für Interrupts. Die Abbildung 3.1 zeigt einen systematischen Überblick über das Steuerwerk und seine Komponenten.

Um zu verdeutlichen, welche Einheit für die Ansteuerung der einzelnen Komponenten verantwortlich ist, wurde eine Farbcodierung eingebunden. In allen Abbildungen stehen blaue Signalnamen für Mikroprogramm-Bits (werden durch das jeweilige Mikroprogramm-Wort gesetzt), grüne kommen aus dem Datenpfad (werden durch Berechnungen oder Multiplexer gesetzt), rote vom Memory-Controller und schwarze von außen. Alle Signale, die aus dem Mikroprogramm kommen, beginnen mit dem Buchstaben „M“. In grau ist die Interrupt-Logik eingezeichnet, mit der der Programmablauf beeinflusst werden kann.

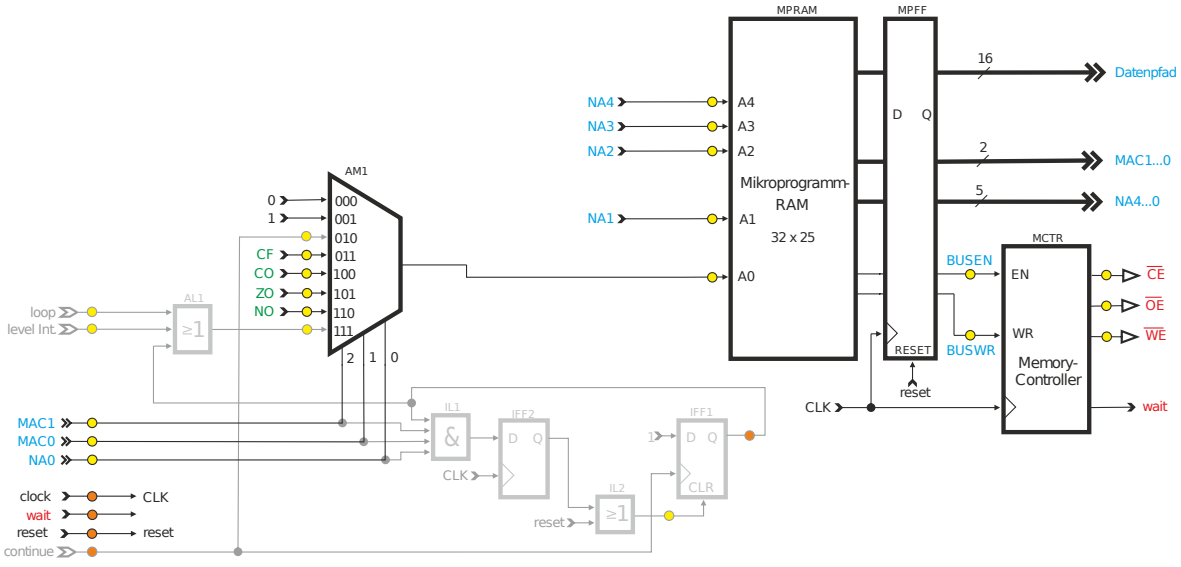


Abbildung 3.1: Schematische Darstellung des Steuerwerks.

## 3.1 Adress-Decodierung

Der Adress-Decoder (Abb. 3.2) ist ein 8-zu-1-Multiplexer, welcher zur Bestimmung des Adressbits A0 dient und welcher direkt vor den Adresseingang geschaltet ist. Dieser Multiplexer wird durch die Mikroprogramm-Bits Microprogram Address Control (MAC0, MAC1) und Next Address 0 (NA0) angesteuert. Die dreistelligen Binärzahlen an den Eingängen des Multiplexers geben an, bei welchem Code an den Steuereingängen 2,1,0 der jeweilige Dateneingang an den Ausgang durchgeschaltet wird.

In Abhängigkeit der Bits MAC0, MAC1, NA0 sind bedingte Verzweigungen bzw. bedingte Sprünge im Mikroprogramm möglich (siehe Tabelle 3.1). Häufig wird als Verzweigungsbedingung ein Flag-Ausgang verwendet. Soll zum Beispiel bei einer negativen Zahl (NO = 1) eine andere Berechnung ausgeführt werden als bei einer positiven Zahl (NO = 0), so muss die nächste Adresse mit einer 0 enden (NA0 = 0) und zusätzlich die Adress-Kontroll-Bits auf 11 (MAC1...0 = 11) gesetzt werden.

In Abhängigkeit des Flag-Werts passt der Adress-Decoder das Bit A0 vor der Ausgabe nach Tabelle 3.1 an. So wird bei einer negativen Zahl der nächste Befehl aus der nächsten ungeraden Adresse geladen, da A0 = NO = 1 gesetzt ist. Bei einer positiven Zahl wird der Wert aus der angegeben Adresse geladen.

Die Signale continue und AL1 sind Bestandteil der Interrupt-Logik (Abschnitt 3.4).

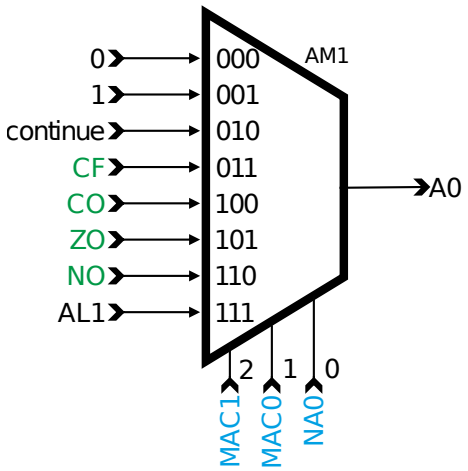


Abbildung 3.2: Schematische Darstellung des Address-Multiplexers.

| Zeile | MAC1 | MAC0 | NA0 | nächste Adresse | | | | | Bemerkung |
|---|---|---|---|---|---|---|---|---|---|
| | | | | Bit A4 | A3 | A2 | A1 | A0 | |
| 1 | 0 | 0 | x | NA4 | NA3 | NA2 | NA1 | NA0 | |
| 2 | 0 | 1 | 0 | NA4 | NA3 | NA2 | NA1 | (INTA) | Level-Interrupt („continue“) |
| 3 | 0 | 1 | 1 | NA4 | NA3 | NA2 | NA1 | CF | Carry Flag des Flag-Registers |
| 4 | 1 | 0 | 0 | NA4 | NA3 | NA2 | NA1 | CO | Carry Out der ALU |
| 5 | 1 | 0 | 1 | NA4 | NA3 | NA2 | NA1 | ZO | Zero Out der ALU |
| 6 | 1 | 1 | 0 | NA4 | NA3 | NA2 | NA1 | NO | Negative Out der ALU |
| 7 | 1 | 1 | 1 | NA4 | NA3 | NA2 | NA1 | (INTB) | Edge-Int. („continue“) oder „loop“ |

Tabelle 3.1: Nächste Adresse im Mikroprogramm.

## 3.2 Mikroprogramm-RAM

In das Mikroprogramm-RAM (Abb. 3.3) müssen alle auszuführenden Mikroprogramm-Wörter gespeichert werden, da aus diesem bei jedem Takt ein neues Wort ausgelesen und an die zugehörigen Steuereingänge angelegt wird. Es können insgesamt 32 Worte zu je 25 Bit abgelegt werden. Die Reihenfolge des Auslesens wird dabei nicht von einem Zähler vorgegeben, sondern vom Steuerwort selbst bestimmt: die sieben niederwertigsten Bits des Steuerwortes dienen dazu, die nächste auszulesende Adresse festzulegen. Mit den fünf Bits Next Address (NA0 bis NA4) wird die nächste auszulesende Adresse angegeben. Das unterste Adressbit (NA0) kann dabei durch einen der Flag-Ausgänge der ALU, durch das zwischengespeicherte Carry-Flag oder durch ein Interrupt-Signal ersetzt werden. Der nachgeschaltete Flip-Flop MPFF setzt durch Drücken des Tasters **reset** die Steuereinheiten zurück (vgl. Tabelle 4.2).

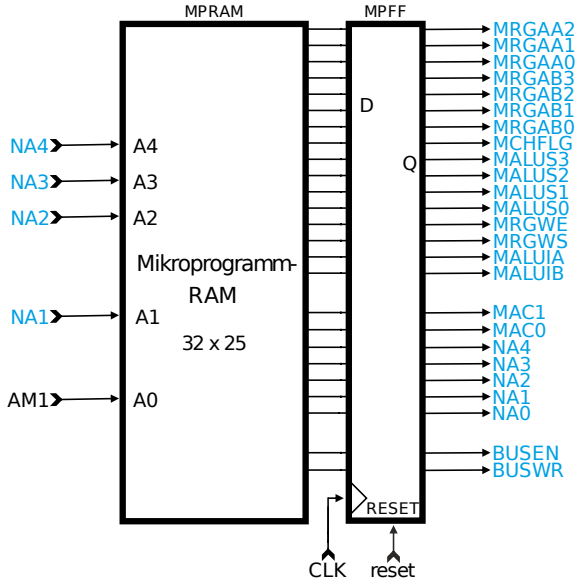


Abbildung 3.3: Schematische Darstellung des Mikroprogramm-RAMs.

## 3.3 Memory-Controller

Der Memory-Controller (Abb. 3.4) bestimmt die invertierten Steuersignale Chip Enable ($\overline{CE}$), Output Enable ($\overline{OE}$) und Write Enable ($\overline{WE}$) für den Daten-RAM. Damit der Daten-RAM richtig angesteuert wird, benötigt jeder Zugriff auf den Daten-RAM-Bus zwei Takte, weshalb der Memory-Controller das Signal „wait“ generiert, welches alle Abläufe im Steuerwerk und dem Datenpfad einfriert, so lange es aktiv ist (jeweils für einen Takt aktiviert).

Ist das eingehende Bit Bus Enable auf 1 gesetzt (BUSEN = 1), so wird der Daten-RAM-Bus angesprochen. Das Signal Bus Write (BUSWR) bestimmt dabei ob gelesen (BUSWR = 0) oder geschrieben (BUSWR = 1) wird.

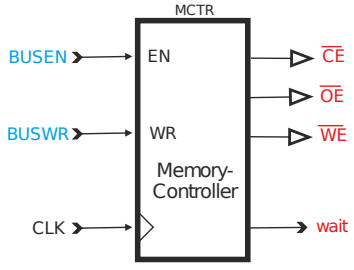


Abbildung 3.4: Schematische Darstellung des Memory-Controllers.

## 3.4 Interrupts

Die Interrupt-Logik (Abb. 3.5) erlaubt es, mit Hilfe von zwei Tastern den Programmablauf des Minirechners 2i manuell zu beeinflussen. Wird der Taster **continue** (kleiner Taster über **step**) gedrückt, so wird das Flip-Flop IFF1 gesetzt und am Eingang 111 des Adressbit-Multiplexers AM1 liegt eine 1 an. Wird dieser Eingang abgefragt, so wird das Flip-Flop IFF1 über IL1, IFF2 und IL2 wieder gelöscht; d.h. das Mikroprogramm sieht pro Tastendruck genau einmal eine 1. Dies kann z.B. verwendet werden, um ein Mikroprogramm per Tastendruck aus einer Warteschleife herauszuholen, um eine (größere) Programmschleife genau einmal abzuarbeiten. Wird dagegen der Taster **loop** (**CONTROL** + **reset**) gedrückt, so liegt am Eingang 111 eine 1 an, so lange der Taster gedrückt wird. Dadurch kann ein Programm in einer Schleife laufen, so lange man den Taster **loop** gedrückt hält.

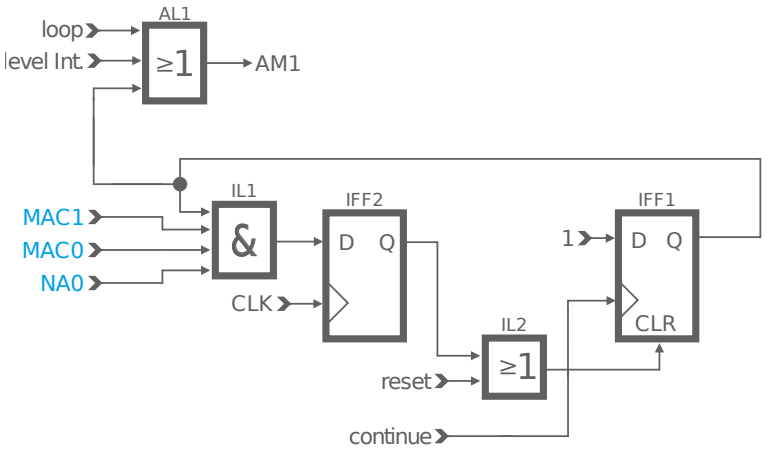


Abbildung 3.5: Schematische Darstellung der Interrupt-Logik.

## 3.5 Befehlsformat

Ein Befehl bzw. ein Mikroprogramm-Wort des Minirechners 2i besteht aus jeweils 25 Bit. Für die Steuerung der ALU, der Multiplexer und der Register im Datenpfad (vlg. Kapitel 2.1 und 2.2) stehen 16 Bits (Bit 0-15) zur Verfügung. Die entsprechenden Bits sind durch ein „M“ am Anfang gekennzeichnet (z.B. MRGWE, Ausnahme: MAC1, MAC0).

Für die Steuerung des Adress-Decodierers, des Mikroprogramm-RAMs und des Memory-Controllers (vgl. Kapitel 3.1, 3.2 und 3.3) stehen insgesamt neun Bits (Bit 16-24) zur Verfü-

| | Steuerwerk | | | | Datenpfad | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Gruppe: | microprogram control | | bus control | | register control | | | | ALU control | | | flag register control |
| Bedeutung: | microprog address control | next address | bus write | bus enable | register address port A | register address port B | register write port select | register write enable | ALU input A select | ALU input B select | ALU function select | change flags |
| Signal: | MAC 1 - 0 | NA 4 - 0 | BUSWR | BUSEN | MRGAA 2 - 0 | MRGAB 3 - 0 | MRGWS | MRGWE | MALUIA | MALUIB | MALUS 3 - 0 | MCHFLG |
| Bit-Nr.: | 24-23 | 22-18 | 17 | 16 | 15-13 | 12-9 | 8 | 7 | 6 | 5 | 4-1 | 0 |

Tabelle 3.2: Aufbau eines Mikroprogramm-Wortes.

gung, welche zum Teil auch für den Zugriff auf das Daten-RAM zuständig sind.

Die Funktionen der einzelnen Bits sind in Tabelle 3.3 dokumentiert, während Tabelle 3.2 den generellen Aufbau der Mikroprogramm-Worte zeigt.

Jedes Mikroprogramm-Wort codiert somit neben den ALU-Funktionen auch den weiteren Programmablauf. Die Herleitung einzelner Befehle wird im Anhang B näher erläutert. Nach dem Herleiten der linken Spalten kann die rechte Seite (ab Spalte MAC) durch die Tabellen 2.1, 3.3 und 3.1 ergänzt werden.

| **Name** | **Anzahl Bits** | **Bit Nr.** | **Bedeutung** |
|---|---|---|---|
| MCHFLG | 1 | 0 | Change Flags: 1 = Ausgänge C,Z,N der ALU in Register übernehmen |
| MALUS3...MALUS0 | 4 | 4...1 | Funktion der ALU |
| MALUIB | 1 | 5 | Auswahl Eingang B der ALU:<br>0 = Register Ausgang B (DOB)<br>1 = Konstante |
| MALUIA | 1 | 6 | Auswahl Eingang A der ALU:<br>0 = Register Ausgang A (DOA)<br>1 = Datenbus (MEMDI: Memory Data In, Input-Register usw.) |
| MRGWE | 1 | 7 | Register Write Enable |
| MRGWS | 1 | 8 | Register Write Select:<br>0 = Write-Adresse ist AA2...AA0<br>1 = Write-Adresse ist AB2...AB0 |
| MRGAB3...MRGAB0 | 4 | 12...9 | Register Adresse Port B und Konstante für Eingang B der ALU<br>xnnn: Register-Adresse = nnn<br>Konstante: 1000 ... 0111 = –8 ... +7 |
| MRGAA2...MRGAA0 | 3 | 15...13 | Register Adresse Port A |
| BUSEN | 1 | 16 | Bus Enable: Datenbus wird angesprochen (read oder write) |
| BUSWR | 1 | 1 7 | Bus Write: Datenrichtung: 0 = lesen, 1 = schreiben |
| NA4...NA0 | 5 | 22...18 | Next Address: nächste Mikroprogramm-Adresse (siehe nächste Tab.) |
| MAC1...MAC0 | 2 | 24...23 | Microprogram Address Control: siehe Tabelle 3 |

Tabelle 3.3: Mikroprogramm-Bits.

# 4 Bedienung

Der Minirechner 2i kennt zwei Betriebsmodi: den Programmiermodus und den Run-Modus. Im Programmiermodus können per Taster Daten in das Mikroprogramm-RAM geschrieben oder dieses ausgelesen werden, oder es kann auf den Daten-RAM-Bus geschrieben oder von diesem gelesen werden. Im Run-Modus wird das Mikroprogramm ausgeführt.

## 4.1 Inbetriebnahme und Einstellungen

Bevor der Minirechner 2i in Betrieb genommen werden kann müssen die Minirechner-Logik und die Taktrate ausgewählt werden. Hierfür müssen auf der Logikplatine (mittlere Platine) der linke Drehregler auf Position 3 (Auswahl des Minirechner 2i) und der rechte Drehschalter auf Position 4 (Auswahl der Taktquelle; 7.3728 MHz) gestellt werden.

Der Minirechner muss mit einer Spannung von mindestens 12 V versorgt werden. Blinken alle LEDs kurz auf, so ist der Minirechner betriebsbereit. Durch das Drücken des Tasters **LOAD** werden die Einstellungen neu geladen. Das heißt vor allem, dass auch die Register gelöscht werden. Der Inhalt des Daten-RAMs bleibt jedoch bestehen.

Hält man die blaue **CONTROL**-Taste gedrückt, so können mit den Tastern des Display-Boards verschiedene Einstellungen vorgenommen werden. Es gelten dann die blauen Beschriftungen. Alle folgenden Ausführungen gelten bei gedrückter **CONTROL**-Taste.

Die Quelle des Grundtaktes für die CPU kann mittels Drehschalter auf der FPGA-Platine gewählt werden. Nach dem Einschalten bzw. Laden der Minirechner-Konfiguration in das FPGA ist als Quelle immer der Quarz am Mikrocontroller (7,37 MHz) gewählt; die Auswahl am Drehschalter wird erst durch Drücken des Tasters **SET** wirksam. Im FPGA gibt es zwei einstellbare Taktteiler, die diesen Grundtakt durch 2 bis $2^{25}$ teilen. Mit den Tastern **slower** bzw. **faster** kann der Takt des aktiven Taktteilers in 25 Stufen langsamer bzw. schneller gestellt werden. Mit der Taste **step 1/4** wird zwischen 1-er oder 4-er Schritten gewählt (rote LED leuchtet bei 4-er-Schritten). Mit der Taste **default** wird ein voreingestellter Wert ausgewählt. Die 4 grünen LEDs an den kleinen Tastern zeigen binär die unteren vier Bit der Nummer der eingestellten Taktrate an (00000 bis 11000, größere Nummer = langsamerer Takt). Die rote LED „clock“ blinkt mit dem eingestellten Takt.

Mit den Tastern **darker** bzw. **brighter** passen Sie die Helligkeit aller LEDs des Displays an, mit **default** wird eine voreingestellte Helligkeit ausgewählt.

Wird der Taster **LED test** gedrückt, so leuchten alle LEDs auf dem Display-Board. Der Taster **LEDs off** schaltet alle LEDs aus, mit dem Taster **LEDs normal** kehrt man in den normalen Betrieb zurück.

## 4.2 Programmiermodus

Die Programmierung des Minirechners erfolgt mittels taktweise auszuführender Befehlswörter. Zum Erstellen solcher Befehlswörter eignet sich die Tabelle in Anhang C. Die Programme können dann per Taster (siehe Tabelle 4.1) zeilenweise in den Mikroprogramm-RAM des Minirechners eingegeben werden. Für diese Eingabe muss jedes Befehlswort zuerst in ein Eingaberegister geschrieben und dann der Inhalt dieses Eingaberegisters in das Mikroprogramm-RAM kopiert werden.

Die Eingabe des Mikroprogramms geschieht im Einzelnen durch folgende Schritte:

1. Schalter **load/run** auf „load“, Schalter **read/write** auf „write“.
2. Mit dem Taster **microprogram** als Ziel das Mikroprogramm-RAM auswählen.
3. Schalter **shift/count** auf „count“ (Adresse hoch- bzw. runterzählen).
4. Mit den Tastern **up** bzw. **down** die Adresse auswählen.
5. Das Mikroprogramm-Wort mit den Tastern **0** und **1** eintippen.
6. Durch Drücken von **store** die Eingabe in das Mikroprogramm-RAM kopieren
7. Nächstes Mikroprogramm-Wort: zurück zu 4.

| **Schalter / Taster** | Funktion |
|---|---|
| **run / step** | — |
| **step** | — |
| **reset** | CPU-Register, Output-Register und UART werden gelöscht. Ausgang und Adresseingang des Mikroprogramm-RAMs werden auf 0 gesetzt. Eingabe-Register, Adressregister und Inhalte von Mikroprogramm- und Daten-RAM bleiben unverändert. |
| **load / run** | Zwischen Programmier- und Run-Modus umschalten. |
| **write / read** | write: Mikroprogramm-RAM bzw. Daten-RAM-Bus sind beschreibbar.<br>read: Schreibschutz (Daten können nur gelesen werden) |
| **store** | write (grüne LED über dem Taster leuchtet): Schreiben des Wertes ins Eingaberegister (LEDs „load: data“) in das Mikroprogramm-RAM bzw. auf den Daten-RAM-Bus. Die grüne LED über dem Taster erlischt, wenn der Taster gedrückt ist.<br>read (grüne LED über dem Taster leuchtet nicht): Erzeugen eines read-Strobe (nur von Bedeutung beim Datenregister des UART). |
| **clear** | Löschen des gesamten Eingaberegisters (LEDs „data“). |
| **0** | 0-Bit von rechts in das Eingaberegister einschieben. |
| **1** | 1-Bit von rechts in das Eingaberegister einschieben. |
| **clear** | Adressregister löschen. |
| **shift / count** | Funktion der beiden Taster rechts auswählen. |
| **0 / down** | Schalter auf „shift“: 0-Bit von rechts in das Adressregister einschieben.<br>Schalter auf „count“: Adressregister um 1 runterzählen. |
| **1 / up** | Schalter auf „shift“: 1-Bit von rechts in das Adressregister einschieben.<br>Schalter auf „count“: Adressregister um 1 hochzählen. |
| **continue** | Setzt beim Drücken das Flip-Flop IFF1 (siehe Steuerwerk) |
| **in FF / in FE** | Auswahl, ob in das Mikroprogramm-RAM oder auf den Bus geschrieben werden soll. Rote LEDs neben den Tastern zeigen die aktuelle Auswahl. |
| **in FD / in FC** | — |

Tabelle 4.1: Tasterbelegung im Programmiermodus (von links nach rechts, vgl. Abb. 2.4).

## 4.3 Run-Modus

Die im Programmiermodus gespeicherten Programme können im Run-Modus ausgeführt werden. Dieser Modus wird durch das Umstellen des Schalters **load/run** auf „run“ eingestellt. Insbesondere können nun auch die Input-Register FC, FD, FE und FF beschrieben werden. Um eines der Register auszuwählen, wird der unter der Beschriftung liegende Taster gedrückt, sodass die entsprechende grüne LED aufleuchtet. Die Register werden dann mit den Tastern **0** und **1** (mittig im Bedienfeld) beschrieben.

Ist der Kippschalter **run/step** auf „step“ gestellt, so kann das Programm mit dem roten **step**-Taster daneben schrittweise ausgeführt werden. Ist stattdessen „run“ eingestellt, so läuft das Programm mit der eingestellten Taktfrequenz. Ist der letzte Programmbefehl mit einem Sprung auf die erste Programmzeile (Adresse 0) versehen, so läuft das Programm in einer Schleife.

Im Run-Modus wird das Eingaberegister „load: data“ in die vier einzelnen Input-Register FC, FD, FE und FF aufgeteilt, die von der CPU unter den Daten-RAM-Adressen FC bis FF (hexadezimal) gelesen werden können. In diesem Modus können die vier Bytes unabhängig beschrieben werden (Auswahl eines Bytes durch die kleinen Taster darunter). Alle Tasterbelegungen (von links nach rechts) sind in Tabelle 4.2 zu finden.

| **Schalter / Taster** | Funktion |
|---|---|
| **run / step** | step: single step: Abarbeiten des Mikroprogramms im Einzelschritt-Modus; run: Mikroprogramm wird fortlaufend abgearbeitet |
| **step** | Wenn Schalter **run/step** auf „step“ steht, wird pro Tastendruck ein einzelner Taktschritt ausgeführt. |
| **reset** | CPU-Register, Output-Register und UART werden gelöscht. Ausgang und Adresseingang des Mikroprogramm-RAMs werden auf 0 gesetzt. Eingabe-Register, Adressregister und Inhalte von Mikroprogramm- und Daten-RAM bleiben unverändert. Das Signal „reset“ (siehe rote LED im oberen Teil) bleibt noch 2 Takte nach dem Loslassen des Tasters aktiv. Im Single-Step-Modus muss danach also zweimal **step** gedrückt werden.<br>Wenn gleichzeitig die Taste **CONTROL** auf dem FPGA-Board gedrückt ist: Eingang 111 des A0-Multiplexers liegt auf high, solange reset gedrückt bleibt (**loop**). |
| **load / run** | Zwischen Programmier- und Run-Modus umschalten. |
| **write / read** | — |
| **store** | — |
| **clear** | Löschen des aktiven Input-Registers (grüne LED leuchtet). |
| **0** | 0-Bit von rechts in das aktive Input-Register einschieben. |
| **1** | 1-Bit von rechts in das aktive Input-Register einschieben. |
| **clear** | — |
| **shift / count** | Funktion der beiden Taster rechts im Programmiermodus auswählen. |
| **0 / down** | Aktives Input-Register um 1 runterzählen. |
| **1 / up** | Aktives Input-Register um 1 hochzählen. |
| **continue** | Setzt beim Drücken das Flip-Flop IFF1 (siehe Steuerwerk) |
| **in FF / in FE / in FD / in FC** | Auswahl, welches der 4 Input-Register mit den Tastern **clear**, **0**, **1**, **down** und **up** angesprochen werden soll. Grüne LEDs neben den Tastern zeigen aktuelle Auswahl an. |

Tabelle 4.2: Tasterbelegung im Run-Modus (von links nach rechts, vgl. Abb. 2.4).

# Anhang

# A Emulator

Der Emulator für den Minirechner 2i bietet eine Möglichkeit selbst geschriebene Programme zu testen. Der von Klemens Schölhorn entwickelte Emulator ist in der Programmiersprache `Rust` geschrieben und ein reines Kommandozeilenprogramm.

## A.1 Installation

Die Installation kann auf zwei Wegen erfolgen. Man kann entweder ein direkt einsetzbares Binary herunterladen, oder man lädt den Quellcode herunter und kompiliert diesen selbst.

**Installation der Binaries**

Für Windows, Linux und MAC sind Binaries des Emulators, also vorkompilierte Programmdateien, verfügbar[1]. Nach dem Download der entsprechenden Archivdatei kann diese in einen beliebigen Ordner entpackt werden. Der Emulator ist dann ohne weitere Installationsschritte direkt verwendbar.

**Download und Kompilieren des Quellcodes**

Der Quellcode ist auf den Git-Servern der Universität Leipzig[2] zu finden. Hier kann das Projekt heruntergeladen oder geklont werden. Das Klonen des Repositorys ist mit folgender Anweisung möglich:

```
git clone https://git.informatik.uni-leipzig.de/ti/hwprak/2i-emulator
```

Die Dokumentation[3] der Git-Server erklärt die Arbeit mit Git genauer. Vor allem der Unterpunkt „Getting started with GitLab“ liefert einen schnellen Einstieg.

Zusätzlich ist die Installation der Programmiersprache `Rust`[4] nötig. Hierbei muss es sich um mindestens Version 1.15.0 handeln. Der Installationspfad muss dann gegebenenfalls manuell der Systemvariable `PATH` hinzugefügt werden. Ist `Rust` bereits mit einer niedrigeren Version installiert kann mittels `rustup` auf eine neuere Version geupdated werden.

Um das Projekt zu kompilieren wechselt man in einer Kommandozeile (Terminal) in den entsprechenden Ordner und führt den Befehl `cargo build --release` aus.

[1] `https://git.informatik.uni-leipzig.de/ti/hwprak/2i-emulator/releases`
[2] `https://git.informatik.uni-leipzig.de/ti/hwprak/2i-emulator` (min. Version 2.0.1)
[3] `https://git.informatik.uni-leipzig.de/help`
[4] `https://www.rust-lang.org/en-US/` (min. Version 1.15.0)

## A.2 Bedienung

**Dateiformat**

Es wird empfohlen die Quelldateien mit der Endung `*.2i` zu speichern. Jede Zeile enthält dabei genau einen Befehl (25-Bit-Mikroprogramm-Wort) und kann optional durch seine Adresse ergänzt werden. Hierzu wird die Adresse, in dem der Befehl steht, am Anfang der Zeile geschrieben und durch einen Doppelpunkt vom eigentlichen Befehl getrennt. Der Programmcode kann außerdem durch Kommentare ergänzt werden. Dafür wird eine Raute (#) als Kennzeichnung verwendet. Innerhalb der Befehle können alle Zeichen außer 0, 1, : und # zur Formatierung verwendet werden.

Das folgende Beispiel zeigt eine mögliche Formatierung der Berechnung des Einerkomplements einer 8-Bit-Zahl:

```
# Berechnung des Einerkomplements (8-Bit)

00000: 00 00001 | 00 | 000 1100 01 | 01 1100 | 0
00001: 00 00010 | 01 | 000 0001 11 | 10 0001 | 0
00010: 00 00011 | 00 | 001 0001 01 | 00 0010 | 0
00011: 00 00100 | 00 | 010 1111 01 | 01 1100 | 0
00100: 00 00000 | 11 | 010 0001 00 | 00 1100 | 0
```

**Befehle und Bedienung**

Der Emulator kann unter Linux mittels `./2i-emulator` in einer Kommandozeile gestartet werden. Unter Windows gelingt dies durch Ausführen der `2i-emulator.exe` durch einen Doppelklick oder in einer Kommandozeile. *Hinweis: Vorher mit der Kommandozeile in den entsprechenden Ordner wechseln. Hat man das Programm selbst kompiliert, so befindet sich die ausführbare Datei im Ordner* `2i-emulator/target/release`.

Eine Programmdatei kann auf zwei Arten ausgeführt werden:

1. Man kann den Pfad beim Starten des Emulators als Argument mit angeben:
   `./2i-emulator pfad/zu/programm.2i` unter Linux und MAC, oder
   `2i-emulator.exe pfad/zu/programm.2i` unter Windows.

2. Wenn der Emulator bereits läuft, kann man den internen `load`-Befehl verwenden:
   `load pfad/zu/programm.2i`

Das Programm kann nun schrittweise (Befehl für Befehl) durch Drücken der Enter-Taste ausgeführt werden. Um die Input-Register zu füllen weist man dem Register einfach den entsprechenden Wert zu (z.B. `FD = 10010110`).

Alle verfügbaren Befehle werden durch den Befehl `help` im Emulator angezeigt. Weitere Informationen sind im Git-Repository des Emulators zu finden.

# B Beispielprogramm

Alle Mikroprogramme des Minirechners 2i weisen die gleiche Grundstruktur auf: Daten aus den Input-Registern einlesen, Daten verarbeiten, Ergebnisse in die Output-Register schreiben. Deshalb ist es sinnvoll das Einlesen der Daten an den ersten und die Ausgabe an die letzten Adressen zu legen. Diese Zeilen müssen dann gegebenenfalls nicht mehr geändert werden. Die restlichen Adressen können für die Datenverarbeitung genutzt werden.

Bevor ein Mikroprogramm-Wort eingegeben wird, sollte vorher der Taster **clear** (mittig im Bedienfeld) gedrückt werden. So braucht man führende Nullen nicht eingeben.

Die Adresse 0 ist der Einstiegspunkt für jedes Programm, dort muss der erste Befehl stehen. Die letzte Zeile des Programms sollte immer einen Sprung zur ersten Zeile beinhalten. So kann das Programm in einer Schleife durchlaufen werden. Dies ist vor allem für das Testen verschiedener Werte nützlich. Hierzu lohnt es sich wie in Kapitel 4.3 beschrieben, den Kippschalter **run/step** auf „run“ zu stellen (so läuft das Programm in einer Schleife). Fällt während des Testens ein Fehler auf, so kann durch Umschalten auf „step“ das Programm schrittweise debugged werden.

Folgendes Beispiel zeigt die Herleitung des Quellcodes für die Berechnung des Einerkomplements einer beliebigen 8-Bit-Zahl. Die hierfür erforderlichen fünf Befehlsworte werden im Folgenden erläutert:

1. **Lade die Konstante FC in das Register R0 (adr. 0)**. Da dazu keine Daten aus dem Memory benötigt werden, wird BUSEN und BUSWR jeweils auf 0 gesetzt. In den Registern werden nicht nur Zwischenergebnisse gespeichert, sondern auch Adressen für den Memory-Bus hinterlegt. Deshalb wird die Hexadezimalzahl FC als Adresse für ein Input-Register in das Register 0 geladen. Dazu genügt es, an MRGAB3…0 die Dualzahl anzulegen, die einem hexadezimalen C entspricht, da die vier höherwertigen Bits der 8-Bit-Zahl mit dem Wert aufgefüllt werden, der an MRGAB3 anliegt. Diese Konstante wird durch den Multiplexer DM2 (MALUIB = 1 = Konstante ausgewählt) an den Eingang B der ALU weitergereicht. Mit der Funktion F = B (MALUS3...0 = 1100) gibt die ALU den Wert unverändert an den Ausgang F weiter. Dieser ist direkt mit dem Dateneingang DI des Registerblocks verbunden. Da durch das Anlegen der Konstante C an MRGAB3...0 die Adresseingänge AB2...0 des Registerblocks nicht mehr frei verwendet werden können, muss die Adresse des Zielregisters über die Adresseingänge AA2...0 angegeben werden (also MRGWS = 0 und MRGAA2...0 = 000). Damit überhaupt in ein Register geschrieben wird, muss noch MRGWE = 1 gesetzt werden.

2. **Schreibe den Wert des Input-Registers FC in R1 (adr. 1)**. Der Wert, der sich an der Memory-Adresse FC (Input-Register) befindet, wird in das Register 1 geladen. Dazu muss der Bus „enabled“ (BUSEN = 1) und im Read-Modus (BUSWR = 0) sein. Durch Auswahl des Registers R0 über die Adresseingänge AA2...0 des Register-Blocks wird erreicht, dass am Ausgang DOA (und damit auf dem Adressbus) die darin gespeicherte Adresse FC anliegt. Da der Bus im Read-Modus ist, wird der Inhalt des dadurch ausgewählten Input-Registers zum Multiplexer DM1 vor Eingang A der ALU „transportiert“. Damit dieser Wert über die ALU ins Register gespeichert werden kann, muss am Multiplexer DM1 der Eingang vom Memory („M“, MALUIA = 1) ausgewählt sein und die ALU die Funktion F = A ausführen. Das Register, in das der Wert geschrieben werden soll, muss über die Adresseingänge AB2...0 ausgewählt werden, da AA2...0 bereits benutzt ist.

3. **Berechne das Einerkomplement (adr. 2)**. Hierfür wird die logische NOR-Verknüpfung verwenden (A NOR A) und das Ergebnis in das Register R1 gespeichert. Das Einerkomplement kann durch eine NOR-Verknüpfung einer Zahl mit sich selbst erreicht werden. Dazu wird über beide Adresseingänge des Registerblocks (AA2...0 und AB2...0) das Register R1 ausgewählt. Mit welchem Adresseingang das zu beschreibende Register ausgewählt wird, ist in diesem Fall egal.

4. **Lade die Konstante FF in das Register R2 (adr. 3)**. Analog zur ersten Zeile wird für die Ausgabe des Ergebnisses die Adresse des Output-Registers FF in das Register R2 geladen.

5. **Schreibe den Wert des Registers R1 in das Output-Register FF (adr. 4)**. Zum Schluss wird der Inhalt des Registers R1 an die Adresse im Memory geschrieben, die im Register R2 steht, also in das Output-Register FF. Dies läuft analog zur zweiten Zeile ab, nur muss hier BUSWR = 1 und MRGWE = 0 gewählt werden. Die letzte Programmzeile hat als nächste Adresse wieder den Anfang des Programms (Adresse 0), was eine Endlosschleife bewirkt. So ist es möglich, das Programm längere Zeit durchlaufen zu lassen.

Da in diesem Programm keine bedingten Verzweigungen (bedingte Sprünge) notwendig sind, wird immer MAC10 = 00 gesetzt und die nächste Adresse (NA) immer um 1 erhöht.

| Adr | Befehl | Steuerwerk | | Bus | Register | | | ALU | | | Flags | Adresse | Steuerwerk | | Bus | | Register | | | | ALU | | | Flags | Bemerkung |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | adr. control | next adr. | func | adr A | adr B | write | in A | in B | funct. | load | | MAC | NA | BUS WR | BUS EN | MRG AA | MRG AB | MRG WS | MRG WE | MALU IA | MALU IB | MALUS | MCH FLG | |
| | | | | | | | | | | | | | 24...23 | 22...18 | 17 | 16 | 15...13 | 12...9 | 8 | 7 | 6 | 5 | 4...1 | 0 | |
| 0 | LD R0, FC | - | 1 | - | 0 | FC | A | - | C | B | - | 00000 | 00 | 00001 | 0 | 0 | 000 | 1100 | 0 | 1 | 0 | 1 | 1100 | 0 | Lade R0 mit der Konstanten FC |
| 1 | LD R1, (R0) | - | 2 | rd | 0 | 1 | B | M | - | A | - | 00001 | 00 | 00010 | 0 | 1 | 000 | 0001 | 1 | 1 | 1 | 0 | 0001 | 0 | Lade R1 mit dem Inhalt der Daten-RAM-Zelle mit Adresse in R0 |
| 2 | COM R1 | - | 3 | - | 1 | 1 | A | R | R | NOR | - | 00010 | 00 | 00011 | 0 | 0 | 001 | 0001 | 0 | 1 | 0 | 0 | 0010 | 0 | Einerkomplement von R1 |
| 3 | LD R2, FF | - | 4 | - | 2 | FF | A | - | C | B | - | 00011 | 00 | 00100 | 0 | 0 | 010 | 1111 | 0 | 1 | 0 | 1 | 1100 | 0 | Lade R2 mit der Konstanten FF |
| 4 | ST (R2), R1 / JMP 0 | - | 0 | wr | 2 | 1 | - | - | R | B | - | 00100 | 00 | 00000 | 1 | 1 | 010 | 0001 | 0 | 0 | 0 | 0 | 1100 | 0 | Speichere R1 in die Daten-RAM-Zelle, deren Adresse in R2 steht |

Tabelle B.1: Bilden des Einerkomplements einer 8-Bit-Zahl.

# C Programmtabelle

Auf der nächsten Seite ist ein Muster der Programmtabelle für das Schreiben eigener Programme zu finden.

In der Spalte Adr. steht die Adresse im Mikroprogramm-Speicher, an die der Befehl geschrieben werden soll. In der Spalte Befehl sollte eine lesbare Abkürzung (mnemonic), wie zum Beispiel *LD FF,Reg 0* (kurz für: lade die Konstante FF in Register 0) stehen. Weitere Spalten sind in Tabelle C.1 erklärt.

| Spalte | Bedeutung |
|---|---|
| **address control** | Signal, mit dem die Adresse des nächsten Befehls modifiziert werden soll |
| **next address** | Adresse des nachfolgenden Befehls (wird ggf. modifiziert). Hinweis: Bei einer bedingten Verzweigung wird der nächste Befehl immer von einer geraden Adresse gelesen, wenn das als Bedingung ausgewählte Signal 0 ist, andernfalls von der folgenden ungeraden Adresse. Trotzdem kann es notwendig sein, als Binärcode für die nächste Adresse (NA) eine ungerade Zahl anzugeben, da Bit 0 (NA0) Teil der Auswahl-Adresse des Multiplexers ist |
| **Bus function** | Modus des Memory-Bus: Lesen (rd), Schreiben (wr) oder nicht ansprechen (-) |
| **Register address A** | Wert am Adresseingang A des Register-Blocks |
| **Reigster address B** | Wert am Adresseingang B des Register-Blocks oder Konstante |
| **Register write** | Adresseingang, der zum Schreiben in ein Register benutzt werden soll (A, B oder - = nicht schreiben) |
| **ALU in A** | Eingang A der ALU: R = Register DOA, M = Memory MEMDI |
| **ALU in B** | Eingang B der ALU: R = Register DOB, C = Konstante |
| **ALU function** | Funktion der ALU |
| **Flags load** | Übernehmen der ALU-Flags in das Flag-Register? (X = ja, - = nein) |

Tabelle C.1: Spalten der linken Hälfte des Mikroprogramm-Worts.

| Adr | Befehl | Steuerwerk | | Bus | Register | | | ALU | | | Flags | Adresse | Steuerwerk | | Bus | | Register | | | | ALU | | | Flags | Bemerkung |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | adr. control | next adr. | func | adr A | adr B | write | in A | in B | funct. | load | | MAC | NA | BUS WR | BUS EN | MRG AA | MRG AB | MRG WS | MRG WE | MALU IA | MALU IB | MALUS | MCH FLG | |
| | | | | | | | | | | | | | 24...23 | 22...18 | 17 | 16 | 15...13 | 12...9 | 8 | 7 | 6 | 5 | 4...1 | 0 | |
| 0 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 1 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 11 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 13 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 14 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 15 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 16 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 17 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 18 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 19 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 20 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 21 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 22 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 23 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 24 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 25 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 26 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 27 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 28 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 29 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 30 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | | | | | | | | | | | | | | | | | | | | | | | | | |