Elektronikpraktikum – Versuch 8: Mikroprozessor

Jonas Wortmann*[1] and Angelo V. Brade†[1]

[1]Rheinische Friedrich-Wilhelms-Universität Bonn

16. September 2024

*s02jwort@uni-bonn.de
†s72abrad@uni-bonn.de

Inhaltsverzeichnis

1 Einleitung **1**

2 Theorie **1**
2.1 Grundlagen 1
2.2 Arithmetic Logic Unit 1
2.3 Akkumulator 1
2.4 Akkumulator mit Datenspeicher 2
2.5 Prozessor 2

3 Voraufgaben **3**

4 Auswertung **5**

5 Fazit **5**

1 Einleitung

In diesem Versuch werden wir einen Prozessor angefangen von einigen wenigen logischen Verknüpfunkt bis zu einem vollständig mit Assambly programmierbaren 8080 Mikroprozessor verstehen. Dafür wird zunächst die Arithmetic Logic Unit und dann dessen Ereweiterungen betrachtet. Zum Schluss führen wir ein Programm zu Multiplikation von Zahlen aus.

2 Theorie

2.1 Grundlagen

Als Grundlage unserer logischen Operationen dient, wie schon zuvor untersucht und verstanden, die Binärarithmetik. Es lassen sich z.B. Zahlen einfach addieren, wobei auf ein Überlauf geachtete werden muss. Ein Überlauf dritt dann auf, wenn das Ergebnis größer, als die Speichergröße der Zahl ist. Es lässt sich auch eine Subtraktion durch umwandlung des Subtrahenden in eine Addition überführen. Dafür wird der Subtrahend bitweise invertiert und von dem gesammten Subtrahenden 1 abgezogen. Im allgemeinen lassen sich mit Binärzahlen aber genauso rechnen, wie wenn man die Dezimalbasis, statt der Binärbasis, benutzt. Neben den Binärzahlen werden auch Hexadezimalzahlen verwendet, da diese sich mit genau 4 Bits darstellen lassen. Durch die verkürtzte schreibweise werden sie zur Datenspeicherung verwendet. So können auch zwei Hexadezimalzahlen einen Byte (8 Bit) bilden.

2.2 Arithmetic Logic Unit

Die Arithmetic Logic Unit (Abb. 1), kurz ALU, ermöglich die Verknüpfung von AND, OR und XOR, sowie die Verknüpfungen des integrierten 8-Bit-Addier-Subtrahierwerks (Abb. 2).

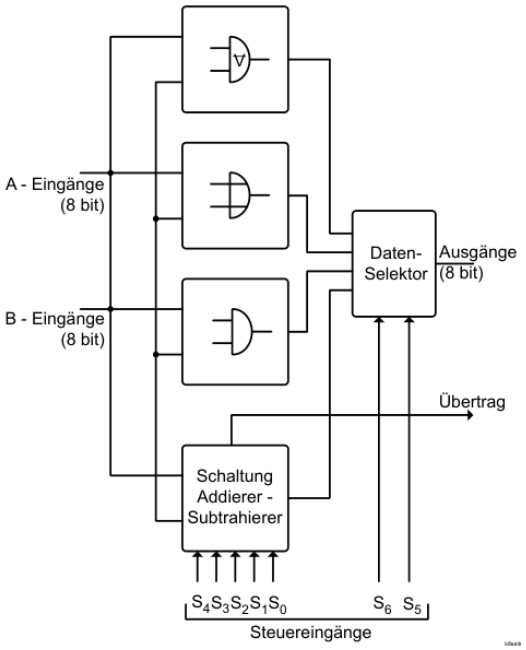

Abbildung 1: Arithmetic Logic Unit

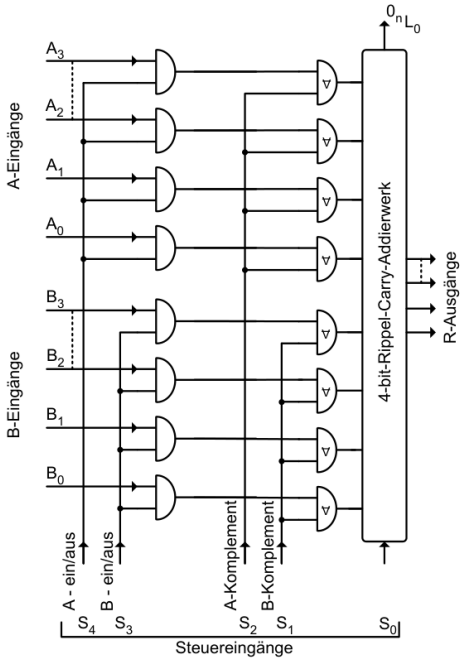

Abbildung 2: 8-bit Addier-Subtrahierwerk

2.3 Akkumulator

Erweitert man den ALU mit einem ROM, sowie einem Register und einer Übertragsflag, so erhält man den Akkumulator (Abb. 3). Der Akkumulator kann mithilfe des Registers Ergebnisse zwischenspeichern. Tritt eine Übertrag

auf, so wird er in der Übertragsflag gespeichert. Der Rom dient der Umkodierung, der Befehle, da von den 32 möglichen Funktionen (Abb. 4) nur 13 (Abb. 5) sinnvoll sind.

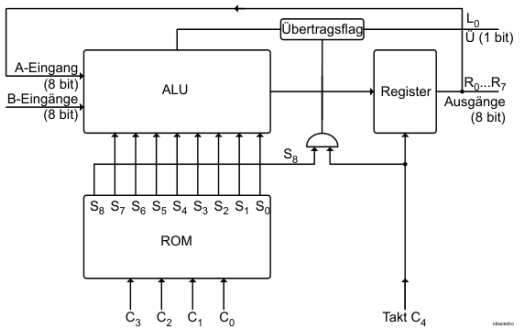

Abbildung 3: Akkumulator

S_4	S_3	S_2	S_1	S_0	Ausgangsfunktic
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	−1
0	0	0	1	1	0
0	0	1	0	0	−1
0	0	1	0	1	0
0	0	1	1	0	−2
0	0	1	1	1	−1
0	1	0	0	0	B
0	1	0	0	1	B+1
0	1	0	1	0	$-B-1 = \overline{B}$
0	1	0	1	1	−B
0	1	1	0	0	B−1
0	1	1	0	1	B
0	1	1	1	0	−B−2
0	1	1	1	1	$-B-1 = \overline{B}$
1	0	0	0	0	A
1	0	0	0	1	A+1
1	0	0	1	0	A−1
1	0	0	1	1	A
1	0	1	0	0	$-A-1 = \overline{A}$
1	0	1	0	1	−A
1	0	1	1	0	−A−2
1	0	1	1	1	$-A-1 = \overline{A}$
1	1	0	0	0	A+B
1	1	0	0	1	A+B+1
1	1	0	1	0	A−B−1
1	1	0	1	1	A−B
1	1	1	0	0	B−A−1
1	1	1	0	1	B−A
1	1	1	1	0	−A−B−2
1	1	1	1	1	−A−B−1

Abbildung 4: 8-bit Addier-Subtrahierwerk Befehle

C_3	C_2	C_1	C_0	Funktion
0	0	0	0	A
0	0	0	1	1
0	0	1	0	$\overline{A}$
0	0	1	1	B
0	1	0	0	0
0	1	0	1	A+1
0	1	1	0	A−1
0	1	1	1	A+B
1	0	0	0	A−B
1	0	0	1	$A \wedge B$
1	0	1	0	$A \vee B$
1	0	1	1	$A\overline{\vee}B$
1	1	0	0	−1
1	1	0	1	
1	1	1	0	
1	1	1	1	

Abbildung 5: Sinnvolle Befehle des ALUs

2.4 Akkumulator mit Datenspeicher

Fügt man dem Akkumulator einen Datenspeicher, hier RAM, hinzu, können jetzt auch zwischenergebnisse nicht nur abgespeichert, sondern auch wieder ausgelesen werden. Dieser ist in Abb. 6 zu sehen.

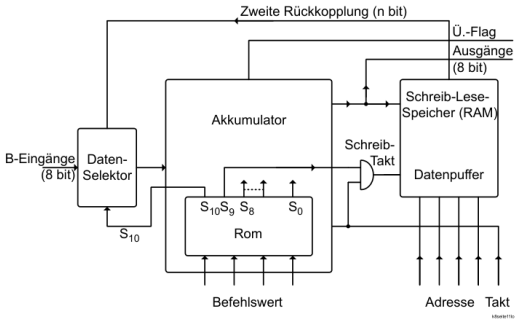

Abbildung 6: Sinnvolle Befehle des ALUs

2.5 Prozessor

Um nun die Konstruktion zu einem Prozessor (Abb. 7) zu vervollständigen, wird ein Programmspeicher mit einem Befehlszähler hinzugefügt. Aus dem Programmspeicher werden mithilfe des Befehlszählers nacheinander die Befehle des Programms aufgerufen. Ist der Befehl ausgeführt, so wird der Befehlszähler um Eins hochgeschalten. So ein Prozessor bildet der Mikroprozessor 8080.

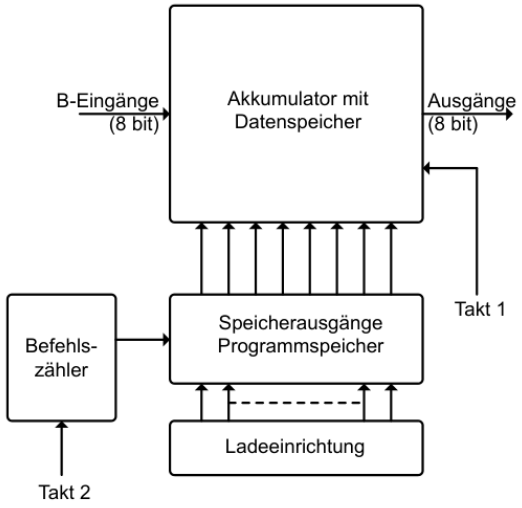

Abbildung 7: Sinnvolle Befehle des ALUs

3 Voraufgaben

A

Die folgenden Dualzahlen, lassen sich in die entsprechenden Hexadezimal- und Dezimalzahlen umwandeln:

Binär	Hexadezimal	Dezimal
1101 1111 0010 1110	DF2E	57134
1111 1111	FF	255

B

Die Zahl 2115_{10} ist auch 100001000011_2 oder 843_{16}.

C

Die Zahl $B75F_{16}$ ist auch 1011011101011111_2 oder 46943_{10}.

D

Es können die folgenden Rechenoperationen durchgeführt weden.

$$\begin{array}{r} 01011011 \\ +01101011 \\ \hline 11000110 \end{array} \qquad \begin{array}{r} 011111111 \\ +000000001 \\ \hline 100000000 \end{array}$$

$$\begin{array}{r} 11000000 \\ -10110101 \\ \hline 00001011 \end{array}$$

Bei Zweierkomplementzahlen gibt die erste Stelle mit 1 an, ob sie negativ, oder mit 0 and, ob sie positiv ist.

Zahl	Vorzeichen
10110111	-
11110000	-
01111111	+
11111111	-

Für die Konvertierung von einer normalen Binärzahl zu einer Zweikomplimentzahl, wird zuerst die Binär zahl invertiert und dann mit 1 addiert. So lassen sich subtraktion mit Zweierkomplementzahlen durch addition darstellen.

$$\begin{array}{r} 11011011 \\ -01101011 \end{array}$$

wird so zu

$$\begin{array}{r} 11011011 \\ +10010101 \\ \hline 101110000 \end{array}$$

Ein Überlauf tritt dann auf, wenn zwei Zahlen addiert werden, wobei das Ergebnis größer ist, als die bereitgestellte Speichergröße. Passiert dies, wird der resultierende zusätzliche Bit nicht gespeichert und es scheint so, als ob die Zahl wieder um 1 plus die maximale Zahl, die in dem Speicher gespeichert werden kann, verringert wird.

Eine Übetrag passiert dann, wenn bei einer Addition der nächste Bit verwendet werden muss. So führt nur die Addition von Zahlen, die an der gleichen Bit-Stelle eine 1 haben zu einem Übertrag.

$$\begin{array}{rrr} 11011111 & 01011011 & 01011011 \\ +00111000 & -10111011\hat{=} & 01000101 \\ \hline 100010111 & & 10100000 \end{array}$$

Bei der ersten Operation entseht ein Überlauf, wobei die Zahl mit einem 9-bit Rechner immernoch mit -23_{10} immernoch im Definitionsbereich eines 8-bit Rechners wäre. Die zweite Operation ist mit -32_{10} hat keinen Überlauf und ist immernoch im Definitionsbereich eines 8-bit Rechners.

Es lassen sich auch Produkte berechnen: 1101 · 1001=1110101.

Genauso auch Quotienten: 1110111/101=10111 mit Rest 100

E

ROM steht für Read-Only-Memory und ist ein Speicher der, wie der Name wage ahnen lässt, nur gelesen werden und nur einmal beschrieben werden kann. RAM steht für Random-Access-Memory und ist ein Speicher, der beliebig gelesen und beschrieben werden kann.

F

Um analoge Signal in Digitale umzuwandeln benötigen wir einen DAC (Digital Analog Converter). Dieser ist durch seine Bitzahl limitiert.

G

Ein ALU hat verschiedene logische Verknüpfungen, wie z.B. AND, OR, XOR und noch grundleregendere, wie z.B. $A + 1$. Erweitert man diesen mit einem Register, einer Übertragungsflagge und einem ROM, so erhält man ein Akkumulator. Mithilfe von einer Taktung kann nun das Ergebniss gespeichert werden und ist somit kein statisches Netzwerk.

H

Erweitert man den Akkumulator mit einem RAM und Programmspeicher, so erhält man einen vollständigen Rechner.

I

Hat ein Rechner ein dedizierten Daten-Bus, Adressen-Bus und Steuer-Bus, so hat er eine sog. Busstruktur. Er rechnet dabei mit Bit in Binärzahlen oder Hexadezimalzahlen für die Befehle.

J

Ein Taktzyklus durchläuft ein Takt der Clock. Ein Befehlszyklus sind alle Takte, die benötigt werden, um einen Befehl auszuführen. Ein Operationszyklus sind alle Takte, die benötigt werdne, um eine Operation durchzuführen.

K

Der 8080-Mikroprozessor kann mit 8 Bits 2^8 also 256 Befehle speichern. Hierbei hat jeder Befehl ein bestimmte Zahl. Diese Zahl wird Operations-Code genannt. Allerdings hat der 8080 nur 115 Befehle.

L

Ein Zweiweg-Tri-state-Datenbus hat nicht nur die Zustände 1 und 0, sondern auch Z, der sog. hochohmige Zustand. Er signalisiert, dass es sich werder noch um 1 oder 0 handelt.

M

Die Länge eines Befehlszyklus hängt davon ab, mit wie vielen Takten der Befehl geholt, decodiert und dann ausgeführt wird. Beim ausführen wird ein Operationszyklus durchlaufen, weshalb der Befehlszyklus auch von ihm abhängt.

N

Der DMA (Direct Memory Access) ermöglicht den Zugriff von Befehlen auf abgespeichert Daten und kann so wiederverwendbare Ergebnisse erreichen, die sonnst erneut berechnet weden müssen. Somit wird weniger Rechenzeit beansprucht.

O

Der Befehlszähler zählt die Befehle und weiß so die Zahl, des als nächstes auszuführenden Befehls. Der Stackpionter speichert die Adressen für abgespeicherte Werte in dem Stack.

P

Der Stack speichert Werte von Größen, die schon zu kompilierzeit bekannt sind. Wie der Name schon suggeriert handelt es sich um ein Stapel an Befehlen. Von diesem Stapel können nur von unten oder oben Befehle hinzugefügt oder entfernt weden. So werden dort Werte die schon beim kompilieren bekannt sind dort gespeichert, sowie Werte dessen benötigte Speichergröße bekannt sind, aber erst beim Ausführen ermittelt werden, dort gespeichert. Während das Programm läuft lässt kein weiterer Speicher mehr reservieren.

Q

Es gibt die volgenden Adressierungsarten: Unmittelbare Adressierung: Der Befehl hat den Wert selber. Direkte Adressierung: Der Befehl kennt direkte Adresse des Werts. Indirekt Adressierung: Der Befehl kennt die Adresse des Werts durch den Stackpointer. Relative Adressierung: Der Befehl kennt durch das Addieren von einer bestimmten Größe die Adresse in Abhängigkeit eines Ausgangpunktes (Arrays auf dem Stack). Indizierte Adressierung: Der Befehl kennt durch Indexen die Adressen einer Reihe an Werten.

R

Der 8080 hat die folgenden Operationszyklen Befehlsaufruf, Speicher lesen, Speicher schreiben, Stack lesen, Stack einschreiben, Eingabe, Ausgabe, Unterbrechung und halten.

S

Der erste Operationszyklus liest den Befehl ein, damit er dann mit dem Befehlregister decodiert werden kann.

4 Auswertung

5 Fazit