Hinweis

Das vorliegende Protokoll wurde im Rahmen der jeweiligen Lehrveranstaltung an der Universität Bonn erstellt. Sofern im oberen Teil der ersten Seite oder auf der unten angegebenen Webseite nicht anders vermerkt, wurde dieses Protokoll von mir, Marvin Zanke, alleine angefertigt und eingereicht. Bei allem in einer anderen Farbe als dem üblichen Blau handelt es sich in der Regel um Korrekturen von mir oder des Tutors. Für mehr Informationen und meine gesamten Unterlagen, siehe:
https://www.physics-and-stuff.com/

Ich erhebe keinen Anspruch auf Richtigkeit und Vollständigkeit des vorliegenden Protokolls! Dies gilt ebenso für obengenannte Korrekturen.

14.09.2016 Versuch 8: Mikroprozessor

In diesem Versuch soll die Funktionsweise eines einfachen Mikroprozessors grundlegend verstanden werden. Die einzelnen Bestandteile, deren Funktionsweise, sowie das Zusammenwirken dieser wird behandelt. Dazu zählen ALU, Akkumulator usw. Der 8080 und Ein-Ausgabe-Einheit von diesem

Theorie:

Multiplexer: Selektionsschaltung, d.h. aus Anzahl von Eingangssignalen kann eins ausgewählt und an den Ausgang durchgeschaltet werden (Drehschalter ist analogon).

Demultiplexer: Gegenstück, d.h. ein Eingangssignal wird auf einen von mehreren Ausgängen geschaltet.

Ripple-Carry-Addierer: Addition mehrstelliger Binärzahlen. Besteht aus n Volladdierern → Zwei n-stellige Binärzahlen können addiert werden; Ergebnis n+1 Stellen.

Subtrahierer: Schaltung aus Operationsverstärkern. Messung von elektrischen Potentialdifferenzen. Am Ausgang: Differenz der beiden Eingangssignale. you are explaining a differential amplifier from Experiment 3 here.

Did they really mean this "Subtrahierer" or does it have something to do with the "adder"?

A "digital" substracter is a logic circuit shown in Fig 8.4 wis $S_4 \sim S_0 = 11011$

ALU: arithmetisches Rechenwerk in Prozessoren (und logische Fkt.) (Addition, Negation, Konjunktion und mehr).

Akkumulator: Register innerhalb CPU, in dem Ergebnisse der ALU gespeichert werden.

Register: Speicherbereiche, mit Recheneinheit verbunden, nehmen Operanden und Ergebnisse aller Berechnungen auf.

Hexadezimal: 0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F

Bus: System zur Datenübertragung zwischen mehreren Teilnehmern über gemeinsamen Übertragungsweg (Steuerung, Adressierung, Übertragung)

Zweierkomplement (-arithmetik): positive und negative Zahlen ohne Vorzeichen speicherbar.
Positive Zahlen mit führender "0" versehen.
Negative Zahlen: alle binären Stellen negiert und zu dem Ergebnis 1 addieren.

BCD-Code: Binary Coded Decimal (0-9)

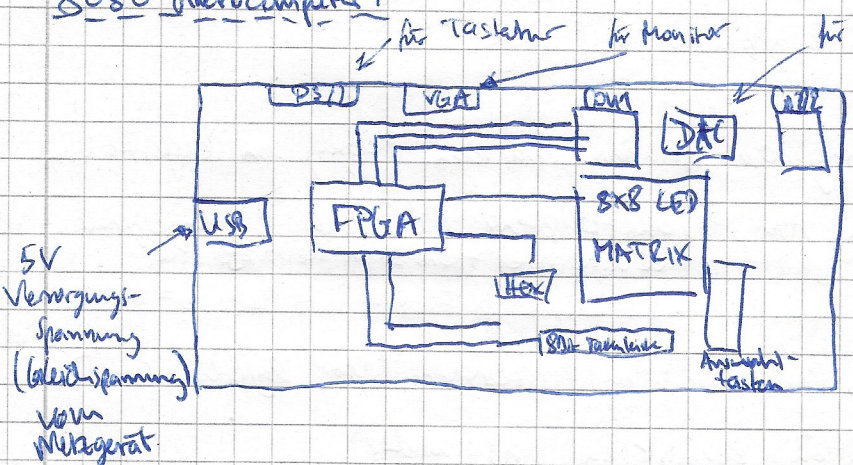
Mnemonics: Zahlenwerte in Maschinensprache schwierig zu merken
⇝ leicht merkbare Kürzel

Maschinensprache: Sprachelemente dieser Programmiersprache entsprechen den vom Prozessor direkt ausführbaren Instruktionen

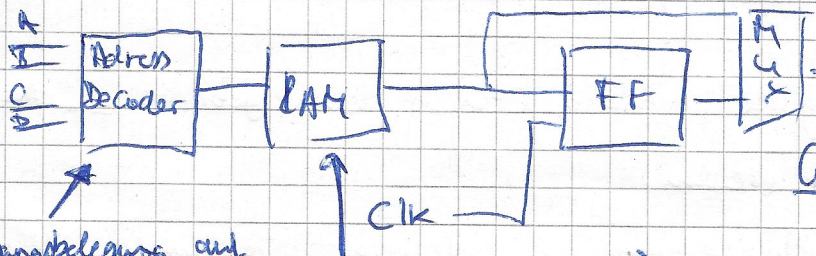
Byte/Bit: 1 Byte ≙ 8 Bit; binary digit

Stapelregister: Adressregister zur Verwaltung von Stacks (Push, Pop)

8080 Microcomputer:

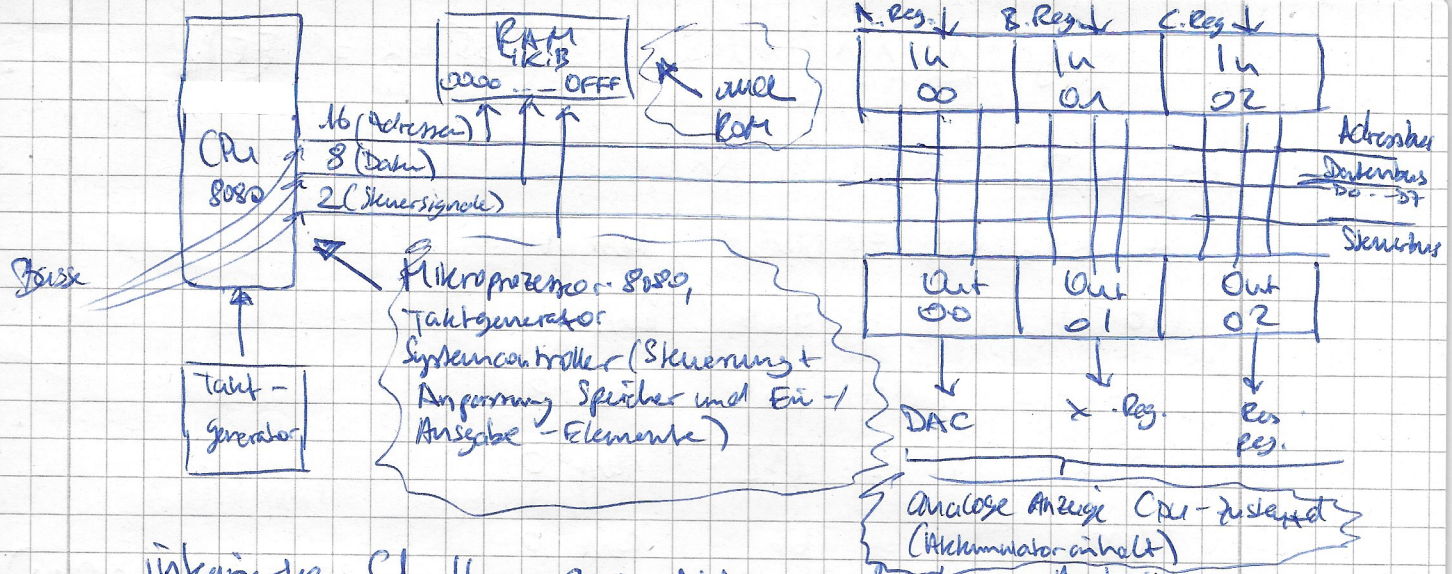


Field-Programmable Gate Array ist das Herzstück; Chip mit frei programmierbaren Logikbaugruppen z.B.



Configurable Logic Block

CLB's können untereinander verbunden werden und komplexere Logikstrukturen



integrierte Schaltung aus ALU, verschiedenen Arbeitsregistern, Ablaufsteuerung (steuert interne Vorgänge logisch und zeitlich).

≙ CPU UND

Programmspeicher, Arbeitsspeicher, Ein-/Ausgabeeinheiten

Taktfrequenz: 24MHz | Systeme: 0: Addierer (Subtrahieren)
1: Kodierte ALU
2: Akkumulator
3: 8 Bit-Rechner mit 8080 Befehlen

Vorausgaben

Aufgabe 1A:

$(1101\ 1111\ 0010\ 1110)_2$

$= 0\cdot 2^0 + 1\cdot 2^1 + 1\cdot 2^2 + 1\cdot 2^3 + 0\cdot 2^4 + 1\cdot 2^5 + 0\cdot 2^6 + 0\cdot 2^7$
$+ 1\cdot 2^8 + 1\cdot 2^9 + 1\cdot 2^{10} + 1\cdot 2^{11} + 1\cdot 2^{12} + 0\cdot 2^{13} + 1\cdot 2^{14} + 1\cdot 2^{15}$

$= (57134)_{10}$

Mit Restwertverfahren folgt:

$(57134)_{10} : 16 \hat{=} 3570$ Rest $14 \hat{=} E$

$(3570)_{10} : 16 \hat{=} 223$ Rest $2 \hat{=} 2$

$(223)_{10} : 16 \hat{=} 13$ Rest $15 \hat{=} F$

$(13)_{10} : 16 \hat{=} 0$ Rest $13 \hat{=} D$

$\hookrightarrow (DF2E)_{16}$

analog: $(1111\ 1111)_2 = (255)_{10} = (FF)_{16}$ ✓

Aufgabe B

$(2115)_{10} : 2 = 1057$ Rest 1
$(1057)_{10} : 2 = 528$ Rest 1
$(528)_{10} : 2 = 264$ Rest 0
$(264)_{10} : 2 = 132$ Rest 0
$(132)_{10} : 2 = 66$ Rest 0
$(66)_{10} : 2 = 33$ Rest 0
$(33)_{10} : 2 = 16$ Rest 1
$(16)_{10} : 2 = 8$ Rest 0
$(8)_{10} : 2 = 4$ Rest 0
$(4)_{10} : 2 = 2$ Rest 0
$(2)_{10} : 2 = 1$ Rest 0
$(1)_{10} : 2 = 0$ Rest 1

$\Rightarrow (1000\ 0100\ 0011)_2$

und $(843)_{16}$

Aufgabe C

$$\begin{array}{r} 01011011 \\ +\ 01110101 \\ \hline 11000110 \end{array} \qquad \begin{array}{r} 11111111 \\ +\ 00000001 \\ \hline 100000000 \end{array}$$ ✓

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

$(10110111)_{(2)} \hat{=} (-73)_{(10)}$ negativ
$(1111\ 0000)_{(2)} \hat{=} (-16)_{(10)}$ negativ
$(01111111)_{(2)} \hat{=} (127)_{(10)}$ positiv
$(11111111)_{(2)} \hat{=} (-1)_{10}$ negative

$$1101\,1101 - 0110\,1011 \longrightarrow 1101\,1101 + 1001\,0101$$

$$1\,0110\,0010 \quad ✓$$

$$\rightsquigarrow 0110\,0010 \hat{=} 98$$

Übertrag $\hat{=}$ carry, also das, was man bei Rechnung mit zur nächsten Stelle nimmt

Überlauf: wenn Zahl nicht mehr korrekt dargestellt werden kann, weil die Anzahl der zur Verfügung stehenden Bits überschritten wurde.

$$1101\,1111 + 0010\,1000 = 1\,0000\,0111$$

Überlauf
carry but not overflow

$$0101\,0011 - 1011\,0011$$

$$\rightarrow 0101\,0011 + 0100\,1101$$

$$1\,0010\,0000$$

~~kein~~ Überlauf because the result is not correct

$$(1101)_2 \cdot (1001)_2 = (1100\,1101)_2$$

$$\begin{array}{r} 1101 \\ 0000 \\ 0000 \\ 1101 \\ \hline 1100\,1101 \end{array} \quad ✓$$

$$(1100\,1111)_2 : (101)_2 = 11011 + \frac{100}{101}$$

$$\begin{array}{l} -(101) \\ \hline 0100\,1 \\ -(\ 101) \\ \hline 0100\,1 \\ -(\ 101) \\ \hline 0100 \end{array}$$

Aufgabe E

ROM: nur lesen, dafür ohne Stromversorgung möglich zu speichern.
RAM: Schreiben (auch rewrite) und lesen. Daten ohne Strom verloren ✓

Aufgabe F

Mit Analog-Digital-Wandler können analoge Signale mit Digitalrechnern verarbeitet werden. Genauigkeit begrenzt durch Abtastfrequenz und Quantisierungsstufen (2^5 Stufen / Messwert bei 5 Bytes) ✓

Aufgabe G:

Arithmetic Logic Unit muss addieren, subtrahieren und logische Operationen durch führen können. Als Akkumulator benötigt der ALU zusätzlich ein Register zum Einspeichern der Ergebnisse + Übertrags-Fl
Wozu dient der Taktpuls?

Aufgabe H

Zusätzlich benötigt man einen RAM, Programmspeicher + programcounter loader

Aufgabe I:

Busstruktur bedeutet, dass die einzelnen Komponenten nicht direkt miteinander verbunden sind, sondern an gemeinsame „Bus" angeschlossen sind (Anzahl gekoppelter Leiter). Digitalrechner arbeiten im Binärsystem. ✓

Aufgabe J:

Taktzyklus ist Frequenz vom Taktgeber, mit der Eingänge der Bauelemente versorgt sind und getaktet werden.
Operationszyklus sind die Takte für Rechen- oder Logikoperationen

Aufgabe K

Da der 8080 (MP) ein 8-Bit Rechner ist, könnte er maximal $2^8 = 256$ Befehle haben. Er besitzt hingegen 244.
Der Operationscode sind die 8-Bit langen Befehle, welche dem Prozessor den Befehlszyklus angeben ✓

Aufgabe L

Der Zweiweg-Tri-State-Datenbus kann Daten in beide Richtungen übertragen. Drei Zustände: HIGH, LOW und nicht gesetzt (HIGH Z) ✓

Aufgabe M:

Länge des Befehlszyklus hängt von Anzahl bzw. Länge der Operationen ab, die ausgeführt werden sollen, sowie der dafür benötigten Takte. ✓

Aufgabe N:

DMA ≙ Direct Memory Access, d.h. Daten werden direkt in den RAM geschrieben (laufen nicht über Programmregister).
Prozessor stark entlastet. Wird für große Datenmengen und deren Transport genutzt, falls sie nicht bearbeitet werden müssen. ✓

Aufgabe O:

Der Befehlszähler durchläuft die Adressen des Programms und arbeitet dieses so ab (Verwaltet Position).
Der Stackpointer zeigt auf den Datenspeicher (letztes Element) und wird bei neuen Werten weitergeschoben (gibt damit Anzahl der Elemente an).

Aufgabe P

Stack ist Datenspeicher, der mit push Informationen aufnehmen kann und mit POP freigeben kann (abrufen). Immer nur der letzte (oberste)

Aufgabe 4

Registeradressierung: Angabe der Registeradresse im Prozessor
→ Daten genutzt.

unmittelbare Adressierung: Adressierung im Befehl enthalten.

absolute Adressierung: Angabe direkte Adresse einer Speicherzelle im Hauptspeicher

indirekte Adressierung: Zwei Adressen → erster Adresswert ≙ Offset auf zweiten Adresswert → Summe ist Adresse im Hauptspeicher

relative Adressierung: Angabe der Speicherzellen als Offset. Dieser wird zum Wert des Spezialregisters hinzuaddiert.

direkte Adressierung: Adresse weist auf Register, in welchem die Adresse des Hauptspeichers steht.

Aufgabe 2:

① Befehlsabruf, ② Speicher lesen
③ Speicher schreiben ④ Output ⑤ Input ⑥ Stackwrite
⑦ Stackread ⑧ Halt ⑨ Interrupt

Aufgabe 5:

Der erste Operationszyklus eines Befehlszyklus tut:

- lädt Befehl aus Speicher in CPU
- Befehlszähler um 1 erhöhen
- Befehlscode in Befehlsregister laden und dekodieren
- Einheiten der Befehlsdurchführung
- Ausführen

Versuchsaufbau und Durchführung

Im ersten Teil schaut man sich das Addier-Subtrahierwerk des Mikrocomputers 8080 an. Dabei ist das Ziel des Versuchs, dieses mit Hilfe von anderen Komponenten zu einem vollständigen Mikrocomputer zu ergänzen. Hier soll nun zuerst eine Funktionstabelle auf ihre Richtigkeit überprüft werden, d.h. ob das A-S-Werk die Steuereingänge S_0 -- S_4 auf den Eingängen A und B richtig verknüpft.

Im zweiten Teil geht es um die Arithmetic Logic Unit, was dem Addier-Subtrahierwerke inklusive einer Schaltung für logische Verknüpfungen entspricht (AND, OR, Exklusiv-OR). Hier schaltet ein Datenselektor die Steuereingänge durch zwischen den 8-Bit Werten der AND, OR, Exkl.OR-Verknüpfung oder dem Ausgang des A-S-Werkes. Es gibt 13 sinnvolle Funktionen, welche mit einem 4-Bit-Steuerwort selektiert werden können.
Man überprüft nun die Schaltung für ein Zahlenpaar und macht einige Überlegungen zum Übertrag.

Im dritten Teil geht es um den Akkumulator für, welchen man die Schaltung um ein Register (Speicher 8 Bit) und einem Flag (1-Bit Speicher) ergänzt. Man kann die Ergebnisse des ALU nun zwischenspeichern. Man subtrahiert so zwei Hexadezimalzahlen und erklärt die Vorgehensweise. Außerdem berechnet man eine Summe im Hex.dez.Syst.

Im vierten Teil geht es um geeignete Steuerkombinationsfolgen mit dieser Schaltung (Prozessorbefehle). Eine Folge von diesen nennt man Computerprogramm. Zusätzlich hat man jetzt einen RAM, um Zwischenergebnisse abzuspeichern. Hier soll man den Sinn und Zweck dieser Erweiterung beschreiben und die prinzipielle

a lots of registers which can be accessed by addressing ⇓ RAM

Difference RAM and Register?

Im fünften und letzten Teil geht es nun um den MP 8080, eine vollständige CPU. Hier überprüft man wieder die Funktion eines Programms, schaut sich einen damit erzeugten Sägezahn an und erzeugt mit einem DAC andere Signale. Außerdem schreibt man ein Programm um zwei 8-Bit Zahlen zu multiplizieren.

Messung

Teil 1

A-Eingang	B-Eingang	C-Register	Result	
7 ≙ 111	8 ≙ 1000	24 ≙ 11000	15 ≙ 1111	A+B
11 ≙ 1011	6 ≙ 110	11011	5 ≙ 101	A-B
—	10 ≙ 1010	01001	11 ≙ 1011	B+1
12 ≙ 1100	/	00010	-1 ≙ 11111111	-A
		10110	-14 ≙ 11110010	-A-1

0000 → 14 = 1110 → 11110001 → 11110010

Am C-Register stellt man ein, wie S0-S4 aussehen, also was mit A und B gerechnet werden soll. Im Resultat steht dann das Ergebnis dieser Operation. Auf dem Display Ausgabe in Hexadezimal und auf den LEDs in Binärsystem

Teil 2

a) A ≙ 01111010 ≙ 122

B ≙ 10000111 ≙ -121

Ā = 10000101

10000110
01111001

C_3	C_2	C_1	C_0	Result	
0	0	0	0	01111010	A
0	0	0	1	00000001	1
0	0	1	0	10000101	Ā
0	0	1	1	10000111	B
0	1	0	0	00000000	0
0	1	0	1	01111101	A+1
0	1	1	0	01111001	A-1
0	1	1	1	00000001	A+B
1	0	0	0	11110001	A-B
1	0	0	1	00000010	A∧B
1	0	1	0	11111111	A∨B
1	1	0	0	11111111	-1
1	1	0	1	—	
1	1	1	0	—	
1	1	1	1	—	

b) Bei Zahlen die die größe der Darstellbaren Zahlen sprengen, benötigt man das Übertragbit. So benötigt man dieses bspw. bei logischen Operationen nicht.

Teil 3:

$(31)_{16} - (8)_{16}$; 00011000 ≙ B
10111000 ≙ A
1000 ≙ C

$= (99)_{16}$
$\hat{=} 153$
$\hat{=} (10011001)_2$

in Zweierkomplement: A ≙ -79
B ≙ 24
-103

Stimmt immer noch, da
$(10011001)_2 \hat{=} -(103)_{10}$

b) Wert in B → in A speichern (und Result)
Neuer Wert B → Add in C → A+B in Result

c) $(22)_{16} - (08)_{16} + (1)_{16} + (10)_{16} = (2b)_{16}$
34 8 1 16 43

Laden in A vom Inhalt B und neue zu speichernde Zahlen immer in B einspeichern. Über C + oder - einstellen und Resultat in A halten.

Teil 4:

RAM zum Zwischenspeichern von mehreren Ergebnissen. Datenselektor selektiert bestimmtes Register in der RAM (input des Rom)

Programmspeicher ermöglicht Programmierung. Speichert Programm als Befehlskette. Befehlszähler zeigt an, welchen Befehl der Programmspeicher ausgeben soll.

Teil 5

a) Signal darstellbar mit Code 3E 00 3C D3 04 C3 02 00

X: 0,1 ms/div

Y: 2V/div

14 kleine Striche

11:39 b) ⇒ 0,28 ms

c)

3E: move to A

00: nichts bewegen (0 auf A)

3C: erhöht A um 1

D3: Output wird erwartet

04: Wert soll ausgegeben werden (DAC)

C3: Springe woanders hin

02: Springe zu 02

07: No operation

11:54 d) B = 0 Periodendauer variabel

11:55 B = 1 DB 01 3D C2 02 00 79 3C D3 04 4F C3 00

e) Mit

11:58 06 08 3E 00 D3 04 4F 80 C2 04 00 79 90 D3 04 C2 0C 00 C3 07

Erzeugen wir ein Rechtecksignal

f) Gemischter Code:

DB 01 47 0E 08 79 A0 C2 0D 00 C3 12 00 3E FF C3

14 00 3E 00 D3 00 C3 00 00

g) Zur Multiplikation zweier 8 Bit Zahlen nutzen wir:

01 47 DB 00 4F 2E 00 26 00 16 00 58

19 79 3D 4F C2 10 00 7D D8 00 7C D3 01 76

Jiko / Simon

Auswertung

Teil1 Wir testen das Addier Subtrahierwerk für einige Ausgangsfunktionen und einige Werte für den A-Eingang und B-Eingang. Diese werden anhand von Schaltern (8 Bits) auf eine "Matrix" mit LED's angelegt, sodass man hier im Binärsystem seine Zahlen in das Gerät eingeben kann. Zusätzlich wird es auf einem Digital display im Hexadezimalsystem angezeigt.
Die gewünschte Ausgangsfunktion stellt man ebenfalls im Binärsystem - am C-Register mit Hilfe von den Bits 0...4 ein. Diese entsprechen dann genau $S_0, \ldots, S_4$ und geben an, welche Operation mit den Eingängen A und B durchgeführt werden soll. Dieses wird im Resultat-Register gespeichert.

A-Eingang		B-Eingang		Result		C-Register	Funktion
dezimal	binär	dezimal	binär	dezimal	binär	binär	-
7	00000111	8	00001000	15	00001111	00011000	A+B
11	00001011	6	00000110	5	00000101	00011011	A-B
-	-	10	00001010	11	00001011	00001001	B+1
-	-	-	-	-1	11111111	00000010	-1
12	00001100	-	-	-14	11110010	00010110	-A-2

Teil2

Im zweiten Teil haben wir uns die Arithmetic Logic Unit angeschaut. Zuerst haben wir für $A = \hat{01111010} \hat{=} 122$

und $B = \hat{10000111} \hat{=} -121$

die Schaltung überprüft. Dazu bemerken wir durch Umkehren der Ziffern auch, dass $\bar{A} = \hat{10000101}$

Dabei geht man analog wie in der vorherigen Aufgabe vor, mit dem einzigen Unterschied, dass mit CLK (Clock) ein Takt ausgeführt werden muss, mit welchem die im Register C liegende Operation ausgeführt wird. Man erhält folgende Tabelle und kann die Funktion der Schaltung damit bestätigen:

A-Eingang		B-Eingang		Result		C3,C2,C1,C0	Funktion
dezimal	*binär*	*dezimal*	*binär*	*dezimal*	*binär*	-	-
122	01111010	-121	10000111	122	01111010	0,0,0,0	A
122	01111010	-121	10000111	1	00000001	0,0,0,1	1
122	01111010	-121	10000111	-123	10000101	0,0,1,0	A(NOT)
122	01111010	-121	10000111	-121	10000111	0,0,1,1	B
122	01111010	-121	10000111	0	00000000	0,1,0,0	0
122	01111010	-121	10000111	123	01111011	0,1,0,1	A+1
122	01111010	-121	10000111	121	01111001	0,1,1,0	A-1
122	01111010	-121	10000111	1	00000001	0,1,1,1	A+B
122	01111010	-121	10000111	-13	11110011	1,0,0,0	A-B
122	01111010	-121	10000111	2	00000010	1,0,0,1	A AND B
122	01111010	-121	10000111	-1	11111111	1,0,1,0	A OR B
122	01111010	-121	10000111	0	00000000	1,0,1,1	A ORNOT B
122	01111010	-121	10000111	-1	11111111	1,1,0,0	-1
122	01111010	-121	10000111	-	-	1,1,0,1	-
122	01111010	-121	10000111	-	-	1,1,1,0	-
122	01111010	-121	10000111	-	-	1,1,1,1	-

Das 8. Bit (S_7) ist dazu da, der Schaltung „mitzuteilen", ob ein Übertrag existiert oder nicht. So benötigt man diesen bei logischen Operationen bspw. nicht und andernfalls würde möglicherweise Information verloren gehen, wenn man arithmetische Operationen durchführt.

√

Teil3

Im dritten Versuchsteil haben wir die Funktionsfähigkeit des Akkumulators erfolgreich überprüft.
Zuerst haben wir die Subtraktion zweier Hexadezimalzahlen durchgeführt:

$(B1)_{16} - (18)_{16} = (99)_{16}$ Dabei war C = 0000 1000,

$(177)_{10} - (24)_{10} = (153)_{10}$ was der Subtraktion entspricht.

A B Result

Im Hexadezimalsystem stimmt die Arithmetik der Schaltung also.

Nun haben wir noch überprüft, ob die Arithmetik richtig bleibt, wenn man die Zahlen im Zweierkomplement interpretiert.
$(177)_{10}$ entspricht dabei $(10110001)_2$ und
$(24)_{10} \hat{=} (00011000)_2$.
Während $(24)_{10}$ also auch im Zweierkomplement $(24)_{10}$ entspricht, wird die 177 wegen dem Überlauf zu $(-79)_{10}$ (Wertebereich: $-128 \ldots 127$).
Interpretiert man die $(153)_{10} \hat{=} (10011001)_2$ nun auch als vorzeichenbehaftete Zweierkomplement-darstellung, so entspricht dies $(-103)_{10}$, in Übereinstimmung zu $(-79-24)_{10} = (-103)_{10}$.

Unsere Vorgehensweise war dabei wie folgt:

- Wähle Register B aus und speichere $(B1)_{16}$ mit den Tastern ein.
- Stelle in C den Load-Befehl ein, welcher den Wert aus B in den Akkumulator und in „A" -Register (Rückkopplung) bringt, wenn man einmal mit CLK taktet.
- Nun speichert man $(18)_{16}$ in das B-Register mit den Tastern und setzt C-Register auf „1000" (Subtraktion).
- Takten mit CLK rechnet „A-B" und speichert das Ergebnis in „Result".

Außerdem haben wir $(22)_{16} - (08)_{16} + (1)_{16} + (10)_{16}$ ausgerechnet, wobei das Resultat $(2B)_{16}$ war.
Rechnet man die analoge Rechnung im Dezimalsystem, so entspräche dies: $(34)_{10} - (8)_{10} + (1)_{10} + (16)_{10} = (43)_{10}$
womit die Ergebnisse übereinstimmen, da $(2B)_{16} \hat{=} 43$.
Hier geht man ähnlich vor, und zwar:

- $(22)_{16}$ ins B-Register speichern
- im C-Register die Load-Funktion bereitstellen und mit Clk takten, s.d. der Inhalt von B auch im A-Register und im Result liegt.
- $(08)_{16}$ ins B-Register speichern und im C-Register „1000" $\hat{=}$ SuB laden, sowie mit Clk takten
  ⇝ in Result liegt nun $(22)_{16} - (08)_{16}$ (damit auch auf A)
- $(1)_{16}$ ins B-Register schreiben und mit „0111" im C-Register (ADD) die Zahlen in A und B mit Clk addieren und in „Result" speichern.
- $(10)_{16}$ in das B-Register schreiben und prüfen, ob noch „0111" (ADD) im C-Register liegt.
- Mit Clk takten und damit wieder das Ergebnis der Rechnung in „Result" schreiben. ✓

Teil 4

Hier sollte man die prinzipielle Wirkungsweise der Schaltungen „Akkumulator plus Datenspeicher" und „Akkumulator plus Programmspeicher" verstehen.
Hier wollen wir nun den Sinn und Zweck der Erweiterungen diskutieren.

Zu Akkumulator plus Datenspeicher

Der RAM (Schreib- und Lesezeichte) erfüllt hier den Zweck eines Speichers, für größere Rechnungen und Operationen um Daten zwischenzuspeichern.
Die einzelnen Register, die wir bis jetzt behandelt und benutzt hatten, lassen kein dynamisches arbeiten mit den gespeicherten Werten zu. So will man die gespeicherte Werte mit dem RAM später zurückholen.
Der Datenselektor hat die Aufgabe, ein bestimmtes Register in der RAM zu selektieren und so dessen Ausgang wieder in den Akkumulator zu laden. Hierbei kann man außerdem wählen zwischen B-Eingang oder Ausgang des RAMs als Eingang für den Akku. Der Input kommt dabei aus der ROM.

✓

Zu Akkumulator plus Programmspeicher

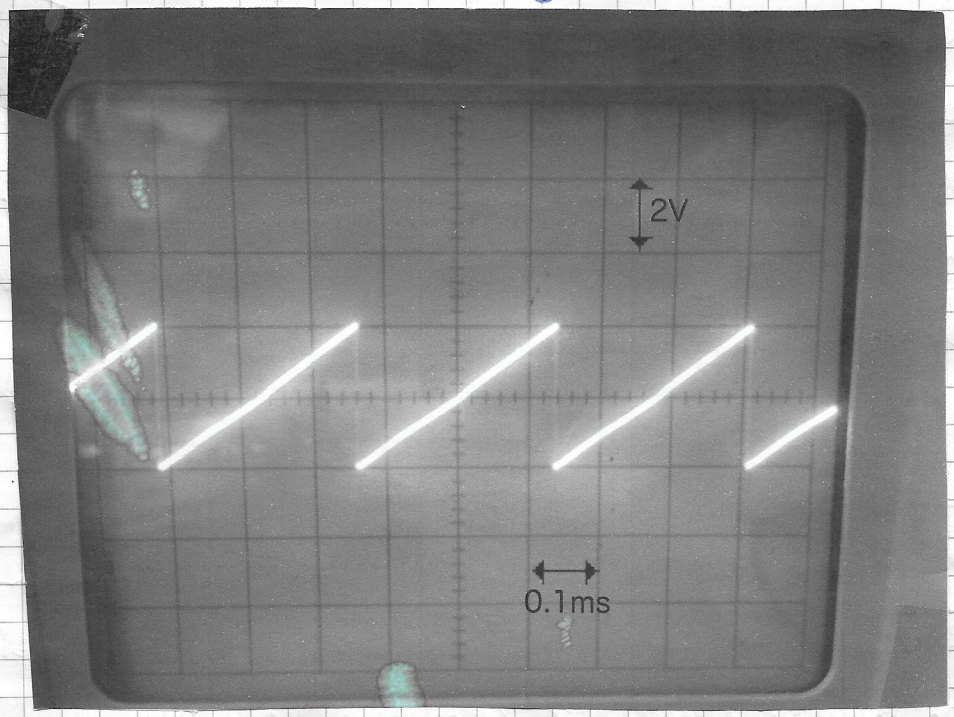
Will man nun, dass das vom Benutzer benötigte Programm selbstständig ausgeführt wird, muss eine vorgegebene Abfolge von Befehlen in beliebiger Reihenfolge zum Abrufen gespeichert sein. Im Programmspeicher wird diese Abfolge von Steuerwörtern gespeichert und geladen.
Der Befehlszähler sorgt während des Ausführens dafür, dass die korrekte Reihenfolge der Befehle abgearbeitet wird. Dieser Befehlszähler gibt genau die Adresse des Befehls an den Programmspeicher, welcher als nächstes auszuführen ist.
Der Programmspeicher gibt dann den Befehl selbst an den Akkumulator zum Ausführen weiter.
Somit können diese Schaltungen bereits selbstständig arbeiten.

✓

Teil5

Im letzten Teil ging es dann darum, mit dem Mikroprozessor 8080 einige Folgen von Befehlen auszuführen und die Resultate mit dem Oszillograph bzw. dem Schaltbrett zu überprüfen.

So haben wir zuerst mittels des im Skript vorgegebenen Programms für einen Sägezahn: 3E 00 3C D3 04 C3 02 00 folgendes Bild aus dem Oszillograph erhalten:



Das digitale Signal des Mikroprozessor wurde dabei mittels einem DAC in das analoge Signal für den Oszillograph umgewandelt.

Für die Periodendauer liest man nun 14 kleine Striche auf dem Oszillogramm ab, was bei der vorgegebenen Skala 0,28ms entspricht. ✓

Für die Funktionen des Programms schaut man in dem gegebenen Befehlssatz des Scripts nach und kommt so zu folgenden Erklärungen:

3E: "move to A", d.h. es soll etwas auf den Speicher A bewegt werden, wobei der nächste Befehl angibt, was dorthin bewegt wird:

00: legt die Adresse 00 auf A.

3C: erhöht den Wert im Register A um 1

D3: bereitet auf ein Output vor; wohin durch nächsten Ausdruck

04: wohin soll ausgegeben werden → 04 ≙ DAC

C3: springe woanders hin (Initialisierung), wohin durch nächsten

02: springe zu 02 ≙ 3C (Loop)

00: no Operation

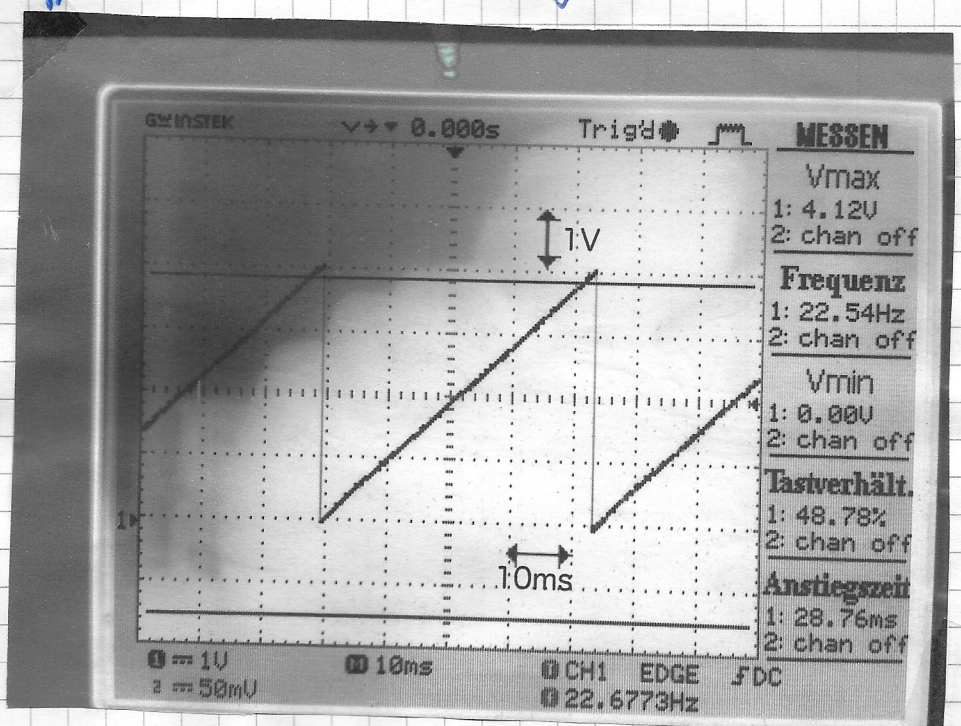
Bei Overflow (A=256) springt der Wert zurück auf 0.

Wir ändern den Programmcode wie folgt ab, um die Periodendauer variabel zu machen. Diese kann man nun über das B-Register eingeben.

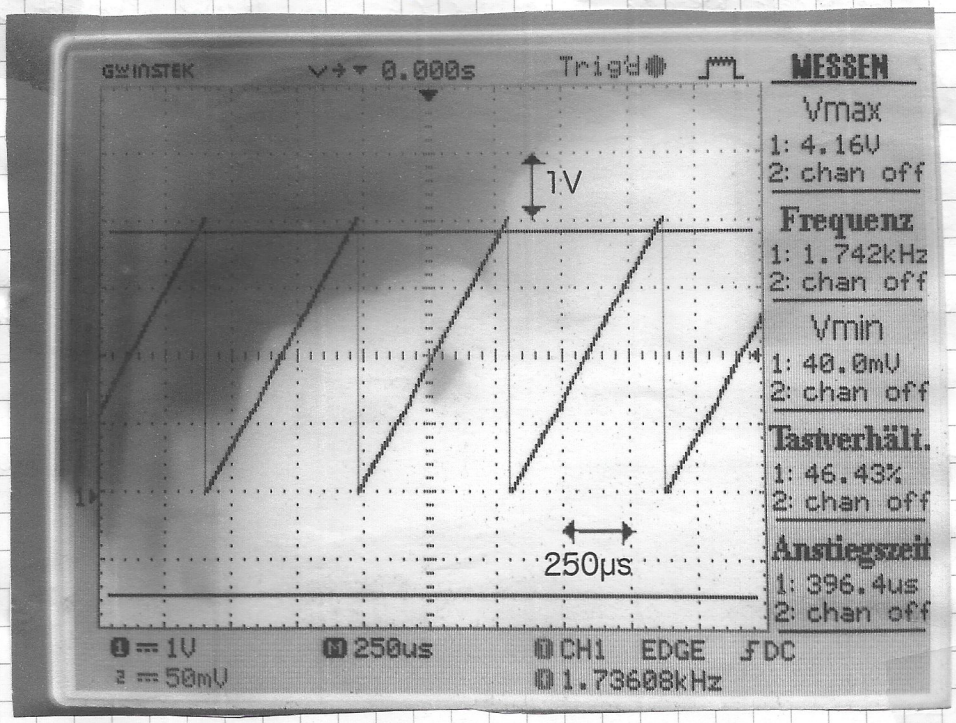
explain your codes →

DB 01 3D C2 02 00 79 3C D3 04 4F C3 00 00

Hier zuerst das Oszillogramm für B=0, wobei die Periodendauer hier offensichtlich 44 ms beträgt.



Nun setzen wir das B-Register auf B=1 und erhalten:

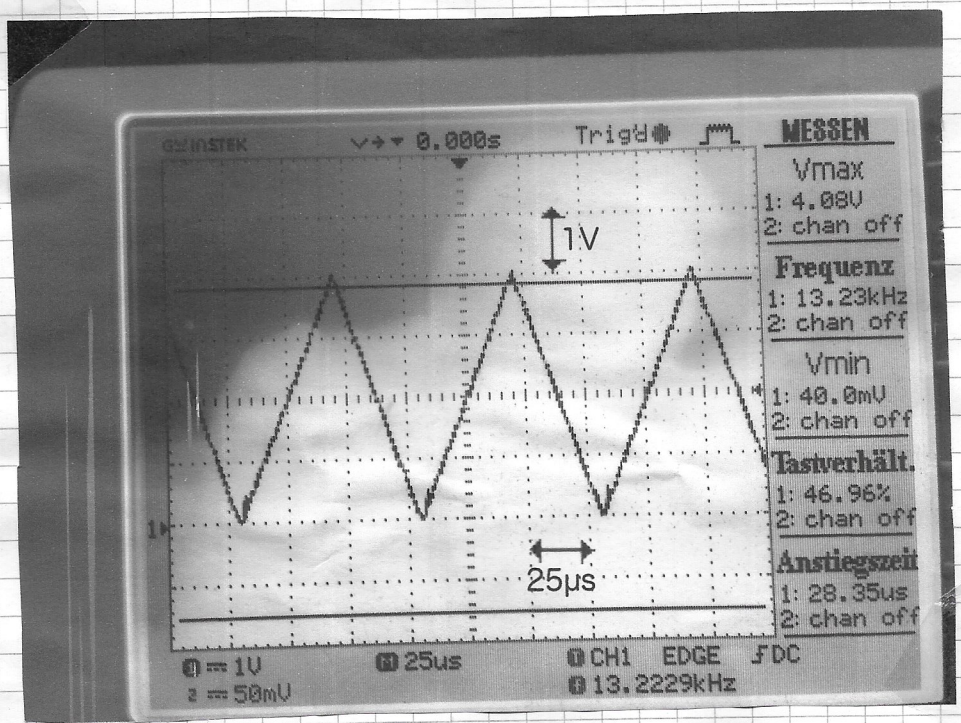


Die Periodendauer beträgt hier 600µs.

Mit Hilfe des Codes: explain your codes

06 08 3E 00 D3 04 4F 80 C2 04 00 79 90
D3 04 C2 0C 00 C3 07 00

erhalten wir außerdem ein Dreiecksignal:



Um zu prüfen, ob das 4te Bit des B-Registers gesetzt ist, kann man folgenden Code verwenden:

explain your codes →

```
D3 01 47 0E 08 79 A0 C2 0D 00 C3 12 00
3E FF C3 14 00 3E 00 D3 00 C3 00 00
```

Ist das Bit mit "1" gesetzt, so leuchtet das gesamte x-Register der Matrix auf.

Will man zwei 8-Bit Zahlen multiplizieren, so eignet sich der folgende Code (Multiplikand und Multiplikator liegen dabei an A-Register und B-Register. Das Ergebnis ist möglicherweise 16-Bit lang und liegt an X und R der LED-Matrix.

explain your codes →

```
D3 01 47 D3 00 4F 2E 00 26 00 16 00 58
19 79 3D 4F C2 10 00 7D D3 00 7C D3 01 76
```

Fazit:

Der Versuch hat - dafür, dass er sehr detailliert in die Materie reingeht und zum genauen Verständnis viel Vorwissen erfordert - erstaunlich gut geklappt. Man konnte, genau wie in Versuch 7, eine Anwendung der Schaltungen aus den ganzen vorigen Versuchen sehen, und hat einen Einblick in die Funktionsweise von Computern erhalten. So wurde einem auch deutlich bewusst, was für eine Pionierarbeit hinter fast all dem steckt, was wir heutzutage als so selbstverständlich sehen - dem Computer. Es war spannend zu sehen, in was für einer Sprache Computer Code auf allen tieferen Niveau interpretieren können und wie alle Bauteile am Ende zusammenwirken.

~~Hero~~ Good
21.09.2016
Joko Hiono