A Einführung

A.1 Geschichte

Warum 8085?

Link: https://www.ordersomewherechaos.com/rosso/fetish/m102/web100/docs/intel-8085-overview.html

Genese: 4004 ⟶ 4040 ⟶ 8008 ⟶ 8080 ⟶ **8085** ⟶ 8086 ⟶ 8088 ⟶ 80186 ⟶ 80286 ⟶ 80386 ⟶ 80486 ⟶ Pentium ⟶ Pentium Pro ⟶ …

Über Registerarchitektur: "We can store data directly inside the CPU. The 8085 has seven 8-bit registers. Each register can store a value from 00h to 0FFh (0 to 255). Unless otherwise specified, a constant which is always numberic is in decimal form. If appended with a character **h** it is assumed to be in hexadecimal form. If a hex constant starts with an alpha-char (A..F) don't forget to include the number **0** in the begining, since that will help the assembler to differentiate between a label and a constant.[1]

The registers are designated by single letters: A, B, C, D, E, H and L. They are not all created equal, and are used for different purposes."

Ein Tutorial, das die Programmierung des 8085 auf einem echten Computer beschreibt (Radioshack's TandyModel 100) findet sich auf: https://www.ordersomewherechaos.com/rosso/fetish/m102/web100/docs/assemb-tutorial.html

A.2 Aufbau einer Zeile:

```
Label:  Befehl Operand ; optionaler Kommentar
```

Label : Name für Speicheradresse. Labels sind optional und können für Variablen oder Sprungziele verwendet werden.

Befehl : Was ist zu tun? Meist auf einige (drei) Buchstaben abgekürzt, sogenannte mnemonische Symbole (Mnemonic)

Operand (= Parameter) : Was sind die Werte (Operanden = Parameter) für den Befehl (0, 1 oder 2 Werte oder Registernamen)?

Kommentar : Wird mit **;** eingeleitet

Jeder Befehl wird vom Assembler in eine Zahl (Opcode) umgewandelt. Die Operanden (= Parameter) werden entweder in den Befehl „hineinkodiert" oder in den nächsten Speicherstellen abgelegt. Im Speicher stehen nur Zahlen ohne Hinweis auf deren Bedeutung (Code oder Daten–> von Neumann-Architektur!).

Tipp: Üblicherweise werden Befehle ohne Label zur besseren Lesbarkeit eingerückt.

A.3 Miniprogramm

a. **Gnusim8085** starten, *Datei / New* erzeugt folgendes Programmgerüst:

```
;<Program title>
       JMP start
       ;data
       ;code
start: NOP
       HLT
```

b. Übersetzen: *Assembler / Assemblieren*

[1] Note that many other programming languages (like Java or C++) denote numbers beginning with a **0** digit as octal numbers!

- Speichern als: **ue1.asm** (Endung muss angegeben werden!)

c. Ergebnis betrachten: *Assembler / Listing* anzeigen

Hinweise:

- Beim Assemblerlisting steht am Anfang der Zeile die Adresse, dann der übersetzte Befehl als Opcode gefolgt von eventuellen Operanden (Werten bzw. Adressen).
- Assembler ist nicht case-sensitive.
- Die Startadresse ist vom Prozessor, einer eventuell vorhandenen Firmware bzw. Betriebssystem abhängig kann man im GNUSim8085 einstellen (Laden bei…)[2].
- Nach einem Label muss (bei diesem Assembler) immer ein Befehl kommen, darum steht in den Beispielen „`start: NOP`“.
- Der Befehl „`HaLT`“ ist für das Beenden der Simulation wichtig, normalerweise läuft ein Prozessor „ewig”.

B Die konkreten Aufgaben

Das Durchführungsprotokoll zur Übung muss folgendes enthalten *(das liefern Sie als Ergebnis Ihrer Arbeit!)*:

- Antworten auf alle gestellten Fragen.
- **Alle** Beispiele als Assemblerlisting eingefügt.
- Einige Fragen sind mit *Internetrecherche* bzw. *Bonusfrage* gekennzeichnet. Diese Fragen sollten erst nach dem Beenden der Übung bzw. während der Ausarbeitung des Protokolls beantwortet werden.
- Einige Beispiele sind als Bonusaufgaben gekennzeichnet. Diese Fragen können vorerst übersprungen werden – Super-Profis lösen aber auch diese Probleme….

Viel Spaß!

B.1 Erstes Programm

Gnusim8085 starten, erstes Programm wie oben.

« Listing hier einfügen, in einer **`Schriftart mit fixer Breite`** formatieren und farblich hervorheben!»

Tipps (Keyboard-Shortcuts):

- Strg-L zeigt **L** isting an (wenn Programm gerade nicht läuft), Alt-F4 schließt Listing-Fenster wieder
- Strg-R (**R** eset) initialisiert alles auf Einschalt-Status (Inhalt des Speichers und der Register wird auf Null gesetzt). An besten immer vor dem Starten eines Assembler-Programmes ausführen.
- F8 assembliert (übersetzt Assembler-Mnemonics in Maschinensprache)
- F5 (oder Pfeil-Icon) führt einen Programmschritt aus

(1) Beantworte folgende Fragen:

a. An welcher Speicherstelle beginnt unser Programm (standardmäßig[3])?

b. Welchen Opcode hat `NOP` (No Operation)?

c. Welchen Opcode hat `JMP`?

d. Wo steht das Sprungziel beim Befehl `JMP`?

e. Wiederholung GINF: Erkläre den Unterschied zwischen „little endian“ und „big endian“!

[2] Mit *Laden bei* akzeptiert der Simulator leider nur Dezimalwerte.
[3] sofern man nicht mittels *Laden bei* einen andere Adresse definiert

f. **Bonusfrage:** Woher kommt dieser Ausdruck?

g. Wie viele verschiedene Befehle (Opcodes) kann es beim 8085 (theoretisch) maximal geben (mit Begründung)?

(2) Starte das Programm mit `Fehlerdiagnose / Schritt in` oder `F5`
   a. Was passiert nach einem weiteren *Schritt in den Code*?
   b. Was steht allgemein im Register `PC`? Wann ändert sich dieser Wert jeweils?
   c. Was ist ein Register eigentlich?

B.2 Erste Befehle

(3) Ergänzen Sie folgende Befehle nach dem Start des Programms (vor dem `HLT`):

```
MVI A, 25h
MVI C, 0EDh
MOV D, A
```

(4) Übersetzen Sie das Programm und führen Sie es schrittweise aus. Beobachten Sie links die Anzeige der Register.

Beantworten Sie folgende Fragen (getrennt für *jeden* der drei Befehle):

   a. Welchen Opcode hat der Befehl?
   b. Was macht der Befehl? Beobachten Sie dazu die Änderungen der Register und Flags und beschreiben Sie diese in eigenen Worten!
   c. Warum gibt es zwei Opcodes für `MVI`?

Tipp: Unter http://web8085.appspot.com/help/index.html findet man eine Beschreibung aller Befehle.

B.3 Variablen

Reservieren von Speicher:

DB *Define Byte* – einzelne Speicherstellen mit Startwert

DS *Define Size* – viele Speicherstellen *(Array of Bytes)*, Größe wird in Byte angegeben

Wichtig Eine von Neumann-CPU kennt nur Speicher, sie unterscheidet nicht zwischen Programm und Variablen. Der gesamte Speicher besteht für die CPU aus Bytes, ob mehrere Bytes (z.B. einen 16- oder 32-bit Wert) ein Array, einen String oder Opcodes darstellen, liegt in der Logik des Programms!

(5) Schreiben Sie folgendes Programm:

```
        JMP start
;data
var1:   DB 195
var2:   DB 2, 66
var3:   DS 8

;code
start:  NOP
        LDA  var2
        INR  A
        STA  var3+3
        HLT
```

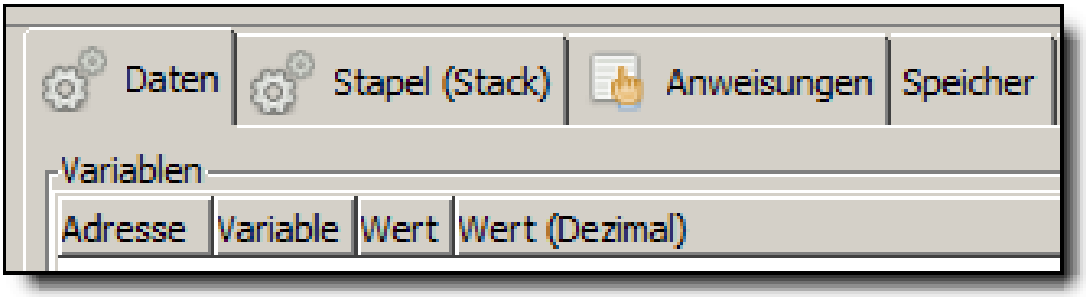

Figure 1: Beispiel Anzeige

(6) Beantworte folgende Fragen

a. Welche Adressen haben die Variablen (rechts unter *Daten*)?

b. Welche Werte stehen im Speicher (rechts unter Speicher) ab der Start-Adresse 4200H? Tipp: Gehen Sie am besten vom Assembler-Listing aus und geben Sie für jede Adresse den Wert sowie eine kurze Beschreibung an!

c. Wie kann man Datenwerte und Befehle („das Programm“) unterscheiden?Im Detail: Wie kann man den Wert bei 4200h (Welcher?) und den Wert in 4203h (welcher?) unterscheiden?

d. Was machen die Befehle `LDA`, `STA` und `INR`? *Tipp:* Den Inhalt der Variablen sieht man rechts unter Daten *bzw. Speicher*!

e. Wann wird `var3+3` ausgerechnet *(siehe dazu auch nächster Punkt)*?

f. **Wiederholung GINF**: Erklären Sie die Begriffe bzw. den Unterschied: „run time“ und „compile time“!

g. Wie viele Taktzyklen dauert die Ausführung des Programms (*ohne* den abschließenden `HLT`-Befehl)? Verwenden Sie dazu die Tabelle aus dem GINF-Unterricht bzw. Handout zur Laborübung!

h. Jemand überschreibt die ersten drei Byte des übersetzten Programmes mit `00`. Was passiert, wenn man anschließend das Programm auf einer 8085 CPU[4] startet?

B.4 Spaß mit Flaggen: Addition

(7) Was macht das folgende Programm? – Ergänzen Sie jede Zeile mit einem Kommentar

```
        JMP start
;data
v1:     DB 25
v2:     DB 100
r:      DB 0

;code
start:  NOP
        LDA  v2
        MOV  B, A
        LDA  v1
        ADD  B
        STA  r

        HLT
```

[4]Mit unserem Simulator kann man dieses Szenario nicht ausprobieren, weil dieser den Prozessor leider nicht perfekt simuliert.

(8) Welcher Wert steht jeweils am Ende in (der Speicherstelle) `r` und im C-Flag *(Carry)*?
   a. Beim Wert `v1 = 25`?
   b. Was passiert, wenn man `v1` auf den Wert `125` ändert?
   c. Was passiert, wenn man `v1` auf den Wert `225` ändert?
   d. **Bonus:** Wie erklären Sie sich dieses eigenartige Verhalten von c?

(9) Was macht folgender Programmteil?

```
MVI A, 025h
MVI B, 011h
STC
ADC B
```

(10) Addieren
   a. Was ist der Unterschied zwischen den Befehlen `ADD` und `ADC`?
   b. Welche Bedeutung hat das *Carry*-Flag (Bit)? Wofür kann man das verwenden?

B.5 Schleife

```
MVI A, 02h
DCR A
DCR A
DCR A
```

(11) Fragen
   a. Was macht `DCR`?
   b. Welcher Wert steht am Ende in A?
   c. Wie erkennt man, dass A den Wert 0 erreicht hat? Welche Flags werden bei `DCR` gesetzt?

(12) Was macht das folgende Programm – ergänzen Sie jede Zeile mit einem Kommentar

```
        MVI A, 03h
loop:   DCR A
        JNZ  loop
```

(13) Schreiben Sie ein Programm, das von 1 bis 8 hinauf zählt.

Tipp: `CMP`, `CPI` *(Compare Immediate)*

B.6 Unterprogramm

(14) Was macht das folgende Programm? Ergänzen Sie wieder jede Zeile mit einem Kommentar!

```
        JMP start

; Unterprogramm
subprog: ADD A
        RET

; Hauptprogramm
start:  NOP

        MVI A, 025h
        CALL subprog
        CALL subprog

        HLT
```

Detailfragen:

a. Untersuchen Sie, was nach dem ersten `CALL` passiert?
   i. Welche Register haben sich geändert?
   ii. Was steht in den Speicheradressen (**Tipp**: Rechts bei *Speicher* ab 0FFF0H oder bei `Stapel (Stack)`schauen!)

b. Was passiert bei `RET`?

c. Was bewirkt das Unterprogramm (und welcher Name wäre daher anstatt des Labels `subprog` besser geeignet)?

d. **Internetrecherche** bzw. Wiederholung GINF: Was ist ein Stack-Speicher? Wozu wird er hier verwendet?

B.7 16-Bit Addition

Gesucht ist ein Programm das zwei 16-Bit Werte addiert

(15) Überlegungen:

a. Wie viele Bytes braucht man für 16 Bit?
b. Wie viele Speicherstellen braucht man für einen 16-bit Wert?
c. In welcher Reihenfolge speichert man diese Werte?
d. Wozu dient noch einmal das *Carry*-Flag?
e. **Bonusaufgabe** (nicht schwierig, wenn man in GINF zugehört hat): Schreiben Sie das Programm!

Hinweise:

- *Tipp*: Wie in der Volksschule! Jeweils eine Stelle (hier ein Byte)addieren – versuchen Sie den Ablauf vorher zu beschreiben und daraus das Programm zu entwickeln.
- Übertrag nicht vergessen, d.h. verwenden Sie `ADD` und/oder `ADC`!
- Das Carry-Flag wird nur von wenigen Befehlen geändert, man kann z.B. Werte aus dem Speicher laden und zurückspeichern ohne dieses Flag zu beeinflussen.

B.8 Indirekte Adressierung

(16) **Internetrecherche** bzw. Wiederholung aus GINF: Was ist ein Zeiger (Pointer)?

Der 8085 verfügt über ein *Pseudo*-Register `M`. Dieses verweist auf die Speicherstelle, deren 16-Bit-Adresse im Registerpaar `H` und `L` steht.

(17) Was macht das folgende Programm? – ergänzen Sie jede Zeile mit einem Kommentar!

a. Welcher Wert steht am Ende in `A` und warum?

```
        JMP start

;data
werte:  DB 2, 3, 5, 7, 11, 0

;code
start:  NOP

        LXI  H, werte
        MOV  A, M
        INX  H
```

```
ADD  M
INX  H
ADD  M

HLT
```

(18) **Bonusaufgabe**: Erweitern Sie das Programm:

a. Addieren Sie in einer Schleife alle Werte bis zum Wert `0`(Endekennung), verwenden Sie ein weiteres Register als Hilfsvariable für die Zwischensumme.

b. MOV ändert keine Flags, verwenden Sie `CMP` oder `CPI` **(Compare Immediate)** oder `SUB` bzw. `SUI` (Subtraktion)

Happy coding!