JOHNNY-Modellrechner

Worum geht es?

Schraubt man einen Personal Computer auf, so erkennt das geübte Auge zwar grob einige wichtige Bestandteile wie den Prozessor, den Speicher (RAM), Festplatten und ähnliches. Aber wie die Maschine tatsächlich arbeitet, davon bekommt man wenig mit: Wenn man den Rechner einschaltet, beginnt sich der CPU-Lüfter zu drehen, und vielleicht beginnen einige Leuchtdioden zu leuchten. Aber was eigentlich in den vielen schwarzen Kästchen passiert, ist nicht zu erkennen.

Trotzdem ist es wichtig, dass wir zumindest eine ungefähre Idee davon haben, was im Computer passiert und was ihn zu dem vielfältig verwendbaren Werkzeug macht, als das wir ihn kennen. Sonst bliebe der Computer für uns so etwas wie ein magischer Gegenstand. Wir werden also versuchen, den Computer zu "entzaubern". Da wir am echten Rechner nur sehr wenig über seine Funktion lernen können, verwenden wir ein Simulationsprogramm mit dem Namen "JOHNNY" (benannt nach dem Computer-Pionier John von Neumann, dem Freunde wegen seiner legendären Partys den Spitznamen "Good Time Johnny" gaben).

Wie geht es?

Viele der nächsten Seiten sind so gemacht, dass Sie selber Dinge mit Hilfe dieses Simulators herausfinden sollen. Es ist daher sehr wichtig, dass der JOHNNY-Simulator beim Arbeiten tatsächlich auch zur Verfügung steht, sei es im Unterricht oder im Selbststudium. Für einige Aufgaben gibt es dabei Lösungs-Vorschläge Diese sollen als Kontrolle die Arbeit erleichtern. Es bringt allerdings wenig, einfach die Lösung einer Aufgabe zu betrachten. Der Lernerfolg stellt sich nur ein, wenn man sich selbst um eine eigene Lösung bemühet, selbst wenn sie am Anfang vielleicht nicht ganz korrekt ist…

Hier lernen Sie ...

1. wie heutige Rechner grundlegend aufgebaut sind.
2. wie man den Modellrechner per Hand steuern kann.
3. die Programmiersprache JOHNNY-Assembler.
4. einfache Assemblerprogramme selbst zu entwerfen und zu testen.
5. wie das Zusammenspiel zwischen Mikro- und Makrobefehlen funktioniert
6. Mikroprogramme für eigene Makro-Befehle zu schreiben.

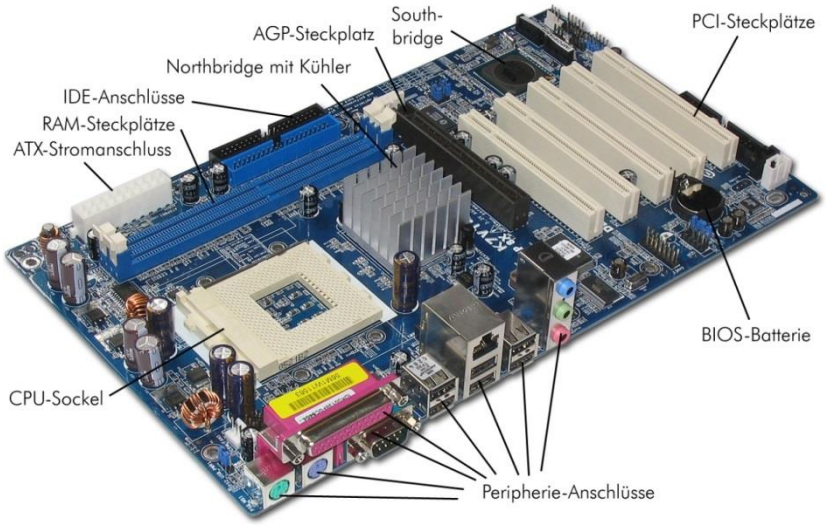
1 Bestandteile eines Computers

1.1 Wenn man einen Rechner (PC) aufschraubt…

Schraubt man einen heutigen PC auf, ergibt sich etwa das Bild rechts:

- Oben links befindet sich das Stromversorgungsgerät.
- Auf der rechten Seite erkennt man die Schächte, in der etwa Festplatten und DVD-Laufwerke montiert werden können.
- Den wichtigste und kompliziertesten Teil sieht man etwa in der Mitte links des Gehäuses: die Hauptplatine (engl.: Mainboard oder Motherboard) mit dem Prozessor (CPU).

Eine solche Platine ist nochmal im nächsten Bild vergrössert gezeigt und die einzelnen Teile beschriftet.

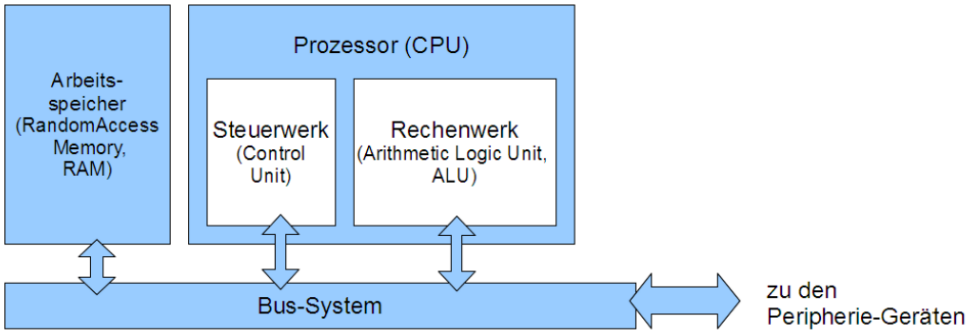


Lassen Sie sich nicht von den vielen Abkürzungen verwirren oder abschrecken, wir werden uns hier nur um diejenigen Dinge kümmern, die für ein generelles Verständnis notwendig sind:

- In der technischen Beschreibung von Rechnern (und natürlich auch der Werbung) wird vor allem auf den Prozessor (CPU, Central Processing Unit) und auf den ...
- ... Arbeitsspeicher (RAM, Random Access Memory) hingewiesen.
- Alle anderen Bestandteile (Festplattenspeicher, Ein- und Ausgabemedien wie USB-Schnittstelle, Tastatur, Maus, Graphik- und Soundkarte usw.) wollen wir hier als Peripherie (also „Umgebung“) bezeichnen.

1.2 Das Bussystem für die Informationsvermittlung

RAM und Peripherie sind über ein so genanntes Bus-System mit der CPU verbunden (Das Wort Bus leitet sich aus dem lateinischen „omnibus“ - „für alle“ ab. Wie beim Strassen-Omnibus wird auch beim Computer-Bus das Transportsystem von verschiedenen Daten als „Verkehrsteilnehmern“ verwendet, obwohl sie unter Umständen unterschiedliche Start- und Zielpunkte haben).



Der Prozessor selbst besteht vor allem aus zwei Einheiten: dem so genannten Rechenwerk (englisch: ALU, Arithmetic Logic Unit), das mathematische und logische Operationen ausführen kann und dem Steuerwerk (englisch: Control Unit), das letztlich alle Abläufe des Rechners steuert.

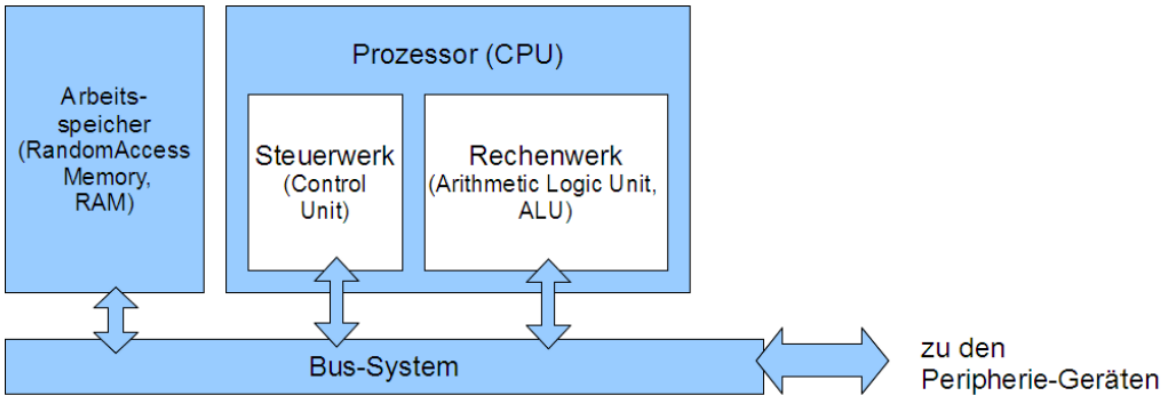
Aufgabe: Computer-Jargon

An vielen Stellen des Textes kommen sowohl Begriffe aus dem Englischen wie auch aus dem Deutschen vorkommen. Das trifft auch für andere Dinge aus der Informatik und Computertechnik zu, die wir hier gar nicht erwähnt hatten.

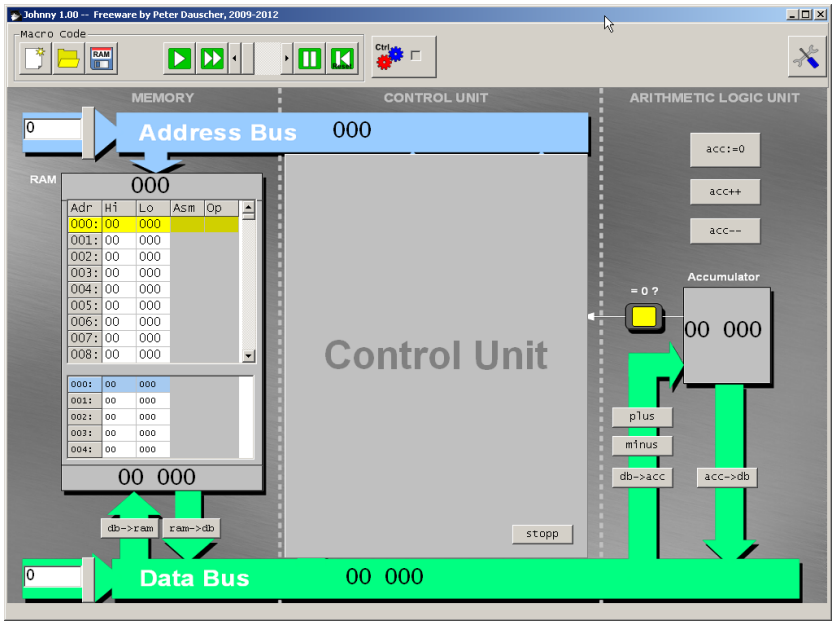
Erstellen Sie eine Übersetzungstabelle von Fachausdrücken englischer Herkunft, mit deutscher Übersetzung – oder umgekehrt.

2 Der Johnny-Simulator

Wie schon gesagt, sieht man einem modernen Rechner seine Funktion nicht an. Um besser zu verstehen, wie Rechner funktionieren, greifen wir deshalb auf eine Simulation zurück. Der Simulator „Johnny“ ist nach dem genialen Entwickler der heutigen Rechnerstruktur, John von Neumann (1903-1957) benannt: Privat war von Neumann für seine Partys berühmt und sein Spitzname war wohl auch deshalb „Good Time Johnny“, worauf der Name des Simulators anspielt. Der Johnny-Simulator konzentriert sich auf CPU mit Rechenwerk und Steuerwerk und den Arbeitsspeicher. Die Benutzungsoberfläche entspricht dabei etwa dem Blockdiagramm im vorhergehenden Abschnitt:



Man erkennt von links nach rechts den Arbeitsspeicher (RAM), das Steuerwerk (Control Unit) und das Rechenwerk (Arithmetic Logic Unit). Das Bus-System des vereinfachten Blockschaltbildes ist in JOHNNYs Benutzeroberfläche in einen Adressbus (oben) und einen Datenbus (unten) aufgespalten worden. Ausserdem kann in den Inhalt des Arbeitsspeichers hineinschauen, ebenso ins Rechenwerk. Beim Start ist der Aufbau des Steuerwerks (Control Unit) noch verdeckt, um die Dinge zunächst einfach zu halten.



Vieles ist beim Johnny-Simulator einfacher als bei einem richtigen Rechner.

- Obwohl praktisch alle "echten" Rechner im Binärsystem rechnen, tut dies der Simulator in unserem gewohnten Dezimalsystem.
- Auch die Funktion des Bus-Systems ist gegenüber echten Rechnern vereinfacht: Normalerweise können sich Bus-Systeme keine Daten merken, sondern sind lediglich Verbindungswege.
- Der Inhalt des RAM kann im Simulator gelöscht, aus einer Datei geladen bzw. in eine Datei geschrieben werden.

All diese Vereinfachungen sollen den Benutzern des Simulators die Bedienung vereinfachen, die Grundprinzipien des Rechners bleiben jedoch erhalten.

3 Download und Installation

3.1 Bezugsquellen

Der Simulator kann kostenlos im Internet heruntergeladen werden, z.B. über eine der folgenden drei Webseiten. Mehr noch: Das Programm ist nicht nur kostenlos, sondern jeder kann auch sehen, wie die Software programmiert wurde. Wer mag, darf das Programm auch verändern und weitergeben, allerdings zu den selben Bedingungen wie das Original.

- Bildungsserver des Landes Rheinland-Pfalz
- Open-Source-Portal Sourceforge
- Download-Server des Heise-Verlags

3.2 Installation

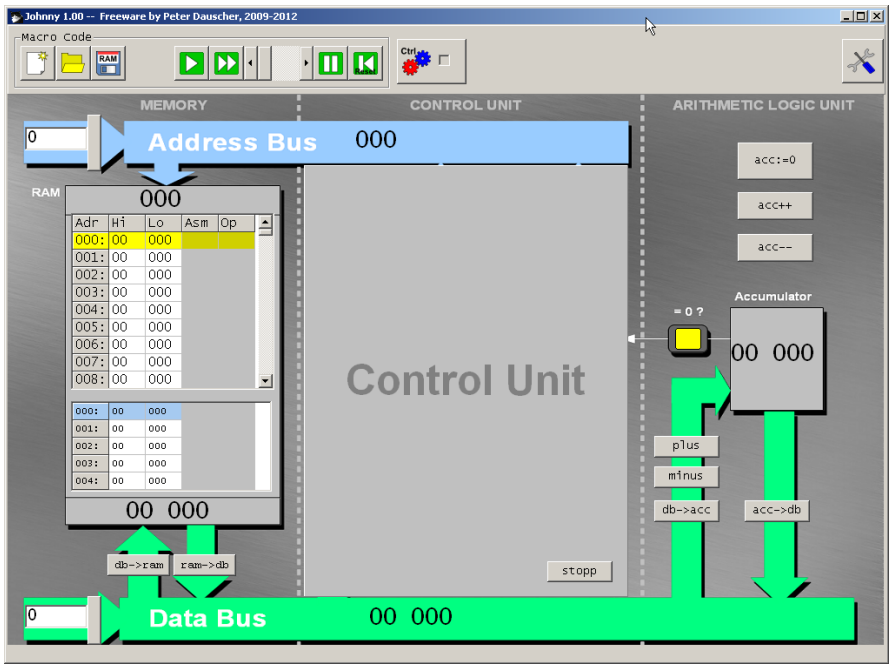
- **Installation unter Microsoft Windows**
  Unter den verschiedenen Microsoft-Windows-Versionen ist eine Installation im engeren Sinne nicht notwendig: Es genügt die gepackte Datei (.zip-Datei) irgendwo auf dem Computer (oder z.B. auch auf einem USB-Stick) zu entpacken.
- **Installation unter anderen Betriebssystemen**
  Bei anderen Betriebssystemen empfiehlt es sich, entweder eine Windows-Systemumgebung zu emulieren oder aber den JOHNNY-Quellcode mit Hilfe der (ebenfalls kostenfreien) Entwicklungsumgebung Lazarus für das jeweilige System zu übersetzen.

3.3 JOHNNY als Open-Source-Projekt

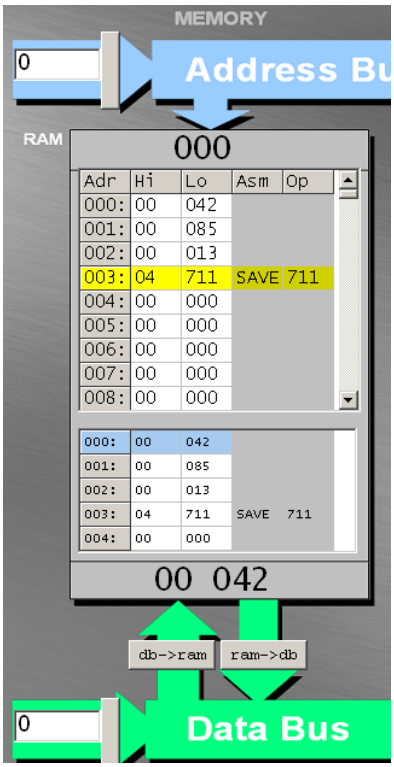
JOHNNY ist ein Open-Source-Projekt. Das bedeutet, dass das Programm nicht nur kostenlos erhältlich ist, sondern jeder kann auch sehen, wie die Software programmiert wurde. Wer mag, darf das Programm auch verändern und weitergeben, allerdings zu den selben Bedingungen wie das Original.

4 Eine erste Erkundungstour

Starten wir JOHNNY nun einfach einmal und betrachten die Benutzeroberfläche.



4.1 Der Arbeitsspeicher (MEMORY)



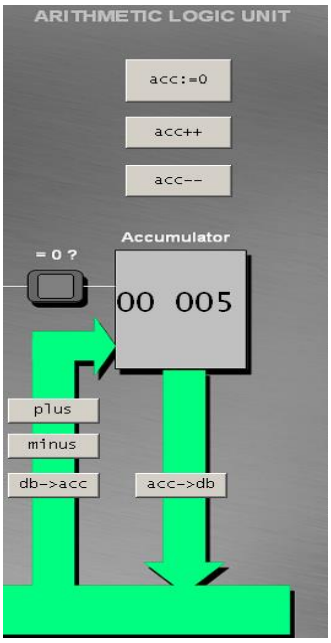
Betrachten wir zunächst den Arbeitsspeicher auf der linken Seite des Bildschirms. Dieser ist als eine Tabelle dargestellt, was seiner tatsächlichen Struktur recht nahe kommt. Die einzelnen Zeilen der Tabelle nennt man auch "Speicherstellen". Die Tatsache, dass man auf jede Speicherstelle einzeln zugreifen kann, gibt dem RAM seinen Namen: "Random Access Memory" kann man (etwas holprig) mit "Beliebiger-Zugriff-Speicher" übersetzen.

- Jede Speicherstelle hat eine Nummer, die so genannte Adresse der Speicherstelle (Spalte „Adr“). Die kann man sich tatsächlich als eine Art "Hausnummer" vorstellen, die klar beschreibt, welche Zeile des Speichers genau gemeint ist.
- In jeder der so durchnummerierten Speicherstellen kann eine Zahl zwischen 0 und 19999 abgelegt werden. Das ist der Inhalt der Speicherstelle. Im Bild rechts steht z.B. in der Speicherstelle 003 der Inhalt 4711. Aus bestimmten Gründen werden dabei die 10000er- und die 1000er-Stelle in einer gesonderten Spalte „Hi“, die 100er-, die 10er- und die 1er-Stelle „Lo“ geschrieben. Die hinteren beiden Spalten „Asm“ und „Op“ werden später eine Rolle spielen, sie sind aber wirklich nur Kommentare. Der eigentliche RAM-Inhalt sind nur die Spalten „Hi“ und „Lo“.

In einem echten Rechner kann man auf die Inhalte der Speicherstelle nur über das Bus-System zugreifen, sie also lesen oder verändern. Dazu muss zunächst die entsprechende Adresse auf den Adressbus gelegt werden (Eingabefeld oben links mit grauer, unbeschrifteter Taste zum Übernehmen des eingegebenen Wertes auf den Adressbus). Dann kann man den Inhalt der Speicherstelle entweder auf den Datenbus kopieren oder aber den Inhalt des Datenbusses in den Speicher kopieren. Beim Simulator kann man direkt in den Speicher hineinschauen. Bei JOHNNY gibt es sozusagen zwei "Fenster", ein oberes und ein unteres. Diese können -- wie im Bild -- den gleichen Bereich von Hausnummern, aber auch unterschiedliche zeigen. Das ist später bequem, muss uns aber im Augenblick noch nicht interessieren. Durch Klicken auf die entsprechende Speicherstelle kann man auch deren Inhalt verändern.

4.2 Das Rechenwerk (ARITHMETIC LOGIC UNIT)

Betrachten wir nun das Rechenwerk (Arithmetic Logic Unit). Diese kann mit Daten, die auf dem Datenbus liegen, rechnen. Kernstück des Rechenwerks ist der so genannte Akkumulator (lat.: „Sammler“). In ihm sammeln sich die Rechenergebnisse an. Drückt man z.B. auf die Taste plus, so wird der Wert auf dem Datenbus zum Wert des Akkumulators addiert und das Ergebnis wieder im Akkumulator gespeichert. Johnny beherrscht von sich aus nur die beiden Rechenoperationen plus und minus (resp. die Addition und die Subtraktion), alles andere muss man ihm sozusagen "beibringen". Der Wert des Akkumulators kann auch wieder auf den Datenbus kopiert werden (mit acc->db). Daneben gibt es noch ein paar andere Möglichkeiten, den Akkumulator zu manipulieren. Versuchen Sie selber heauszufinden, welche dies sind und was deren Funktion ist. Wichtig für später ist auch die Kontrolllampe links neben dem Akkumulator. Diese zeigt an, ob der Inhalt des Akkumulators gerade den Wert 0 hat.



Aufgabe 1

Versuchen Sie, in die Speicherstelle 5 den Wert 42 zu schreiben. Versuchen Sie es einmal mit direktem Anklicken der Speicherstelle (was so natürlich nur in einem Simulator funktioniert) und einmal über den Adress- und den Datenbus.

Aufgabe 2

In dem Schema des Simulators sind (grau) verschiedene Tasten mit Beschriftungen. Finden Sie heraus, welche Funktion sie haben. Fertigen Sie dazu eine übersichtliche Tabelle an.

Aufgabe 3

Speicheren Sie in Speicherzelle 5 den Wert 42 und in Speicherzelle 6 den Wert 23. Versuchen Sie, nur durch Eingabe von Speicheradressen und Klicken auf die verschiedenen Tasten die beiden Werte zu addieren und das Ergebnis in der Speicherzelle 8 zu speichern. Notieren Sie die Reihenfolge Ihrer Eingaben bzw. Tastendrücken.

Eine erste Erkundungstour -- Lösung A.1

Belegung der Speicherstelle über das Bus-System

- Zunächst die Adresse 5 auf den Adressbus legen.
  Dafür muss man die Zahl in das Eingabefeld oben links schreiben und dann entweder die graue, schmale Taste in der Pfeilspitze drücken oder wahlweise auf die Eingabetaste (ENTER) der Tastatur drücken.
- Dann die Zahl 42 auf den Datenbus legen.
  Dafür muss man die Zahl in das Eingabefeld unten links schreiben und dann entweder die graue, schmale Taste in der Pfeilspitze drücken oder wahlweise auf die Eingabetaste (ENTER) der Tastatur drücken.
- Nun die Taste

  db->ram

  drücken.

  Jetzt müsste man in der Speicherzelle Nr. 5 den Wert 00.042 sehen.

Eine erste Erkundungstour -- Lösung A.2

Taste	Funktion
db->ram	Kopiert den Inhalt des Datenbusses in eine Speicherstelle des RAM, und zwar in diejenige, deren Adresse auf dem Adressbus liegt.
ram->db	Kopiert den Inhalt einer Speicherstelle auf den Datenbus, und zwar aus derjenigen Speicherstelle, deren Adresse auf dem Adressbus liegt.
db->acc	Kopiert den Inhalt des Datenbusses in den Akkumulator.
acc->db	Kopiert den Inhalt des Akkumulators auf den Datenbus.
plus	Addiert den Inhalt des Datenbusses zum Inhalt des Akkumulators hinzu.
minus	Subtrahiert den Inhalt des Datenbusses vom Inhalt des Akkumulators.
acc:=0	Setzt den Inhalt des Akkumulators auf den Wert 0.
acc++	Addiert zum Inhalt des Akkumulators den Wert 1 hinzu.
acc--	Subtrahiert vom Inhalt des Akkumulators den Wert 1.

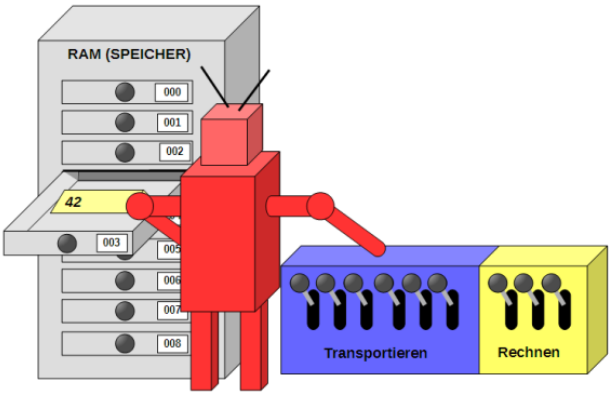
Eine erste Erkundungstour -- Lösung A.3

- Adresse 5 auf den Adressbus legen
- Taste `ram->db`
- Taste `db->acc`
- Adresse 6 auf den Adressbus legen
- Taste `ram->db`
- Taste `plus`
- Adresse 8 auf den Adressbus legen
- Taste `acc->db`
- Taste `db->ram`

5 Makrobefehle im Speicher

Was genau haben wir eigentlich im letzten Abschnitt gemacht?
Wir haben Adressen eingetragen und Schalter für einfachste Operationen zum Transportieren von Daten und zum Rechnen betätigt. Auf diese Weise ist es uns gelungen, den Inhalt des Speichers zu ändern.

Versuchen wir, uns das anschaulich vorzustellen. Der Speicher ist eine Art Aktenschrank mit Schubladen, in denen jeweils eine Zahl liegt. Das Eintragen von Adressen entspricht dem Aufziehen der jeweiligen Schublade. Das Drücken der Schalter entspricht kleinsten Befehlen, die unser Computer versteht. Man nennt diese Befehle deshalb auch Mikrobefehle. Manche davon sorgen für den Transport von Daten von einer Stelle zur anderen, manche anderen veranlassen Rechenoperationen.



Wenn wir ihn so verwenden, benimmt sich der JOHNNY-Simulator allerdings noch nicht wie ein richtiger Computer, wie wir ihn kennen: Bisher müssen wir selbst sozusagen noch die Schubladen aufziehen und die Hebel bedienen: Wir spielen die Rolle des roten Roboters in der Graphik noch selbst. Richtige Computer arbeiten Computerprogramme hingegen automatisch ab, die im Rechner irgendwo gespeichert sind.

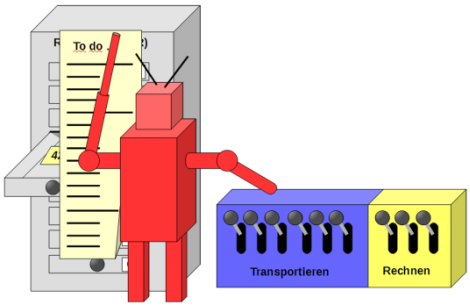
Hierfür müssen wir zunächst zwei Probleme lösen:

- Wir müssen es schaffen, dass Befehle und Folgen von Befehlen im Speicher des Rechners gespeichert sind.
- Und zwar idealerweise in einer Form, die nicht allzuviel Speicherplatz verbraucht.

Wir beginnen im nächsten Abschnitt mit der zweiten Forderung, weil wir so gleich eine gute Idee für die erste Forderung bekommen können.

5.1 Abkürzungen für Befehlsfolgen

Wir beginnen mit einem Gedankenexperiment: Wir möchten jemandem anderen mitteilen, wie er den JOHNNY-Simulator bedienen muss, um eine bestimmte Rechnung damit genauso auszuführen, wie wir das wollen. Dafür schreiben wir ihm eine sehr sehr detaillierte (und damit auch lange) Anweisung bestehend aus den Einzelschritten, die er durchzuführen hat.



Aufgabe 1: Eine kompliziertere Rechnung

Notieren Sie auf Papier oder in einer Textdatei eine Folge von Operationen, die folgendes tut: Unser Modellcomputer soll ....

- ... die Zahl in Speicherstelle 10 nehmen ...
- ... und die Zahlen in den Speicherstellen 11 und 12 dazu addieren.
- Danach soll der die Zahl in Speicherstelle 14 davon subtrahieren.
- Das Ergebnis soll in der Speicherstelle mit der Adresse 15 abgespeichert werden.

Neben den Mikrobefehlen (Tastendrücken) selbst wie

```
ram->db
```

brauchen wir zusätzlich noch eine Anweisung wie

```
Lege Adresse ... auf den Adressbus
```

wobei "..." für die entsprechende Adresse, also die Nummer der Zeile im RAM steht.

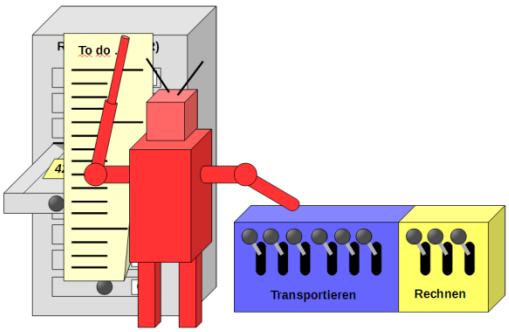
Zählen Sie auch die Anzahl der einzelnen Befehle, die für die Ausführung des gesamten Rechenweges notwendig sind.

Aufgabe 2: Gliederung einer Befehlsfolge

Teilen Sie nun die Befehlsfolge aus Aufgabe 1 so in Abschnitte auf, dass die einzelnen Abschnitte sinnvoll der jeweiligen Abkürzungen entsprechen. Ordnen Sie dabei jedem Abschnitt übersichtlich die entsprechende Abkürzung zu:

```
TAKE 10
ADD  11
ADD  12
SUB  14
SAVE 15
```

5.2 Makrobefehle



Abkürzungen wie "TAKE 10" für ganze Mikrobefehlsfolgen nennt man Makrobefehle, manchmal auch Maschinen- oder Assembler-Befehle. Sie bestehen im allgemeinen aus dem Befehlsteil (hier "TAKE") und dem Adressteil (hier "10"), der beschreibt, auf welche Speicherstelle sich die Operation bezieht. Würde man uns ein solches Programm zur Ausführung geben, könnten wir uns vorher auf einer Tafel notieren, wie die einzelnen Makrobefehle in einzelne Mikrobefehle zu übersetzen sind, damit wir nicht jedes Mal nachdenken müssen.

Aufgabe 3: Achtung - Verwechslungsgefahr!

Jemand behauptet:
"Der Befehl `TAKE 42` *bedeutet, dass der Computer die Zahl 42 in den Akkumulator kopieren soll."*

Nehmen Sie (begründet) Stellung zu dieser Behauptung…

Aufgabe 4: Vorteile von Makroprogrammen

Diskutieren Sie, weshalb die Verwendung von Abkürzungen helfen kann, ...

- Programme kleiner und damit speicher-schonender und leichter eingebbar zu machen
- Programme für den Programmierer übersichtlicher und verständlicher zu machen.

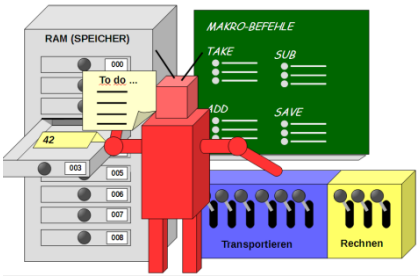
Abkürzungen für Befehlsfolgen -- Lösung A.2

Abschnitt	Abkürzung
• Lege Adresse 10 auf den Adressbus • ram->db • db->acc	TAKE 010
• Lege Adresse 11 auf den Adressbus • ram->db • plus	ADD 011
• Lege Adresse 12 auf den Adressbus • ram->db • plus	ADD 012
• Lege Adresse 14 auf den Adressbus • ram->db • minus	SUB 014
• Lege Adresse 15 auf den Adressbus • acc->db • db->ram	SAVE 015

6 Der Speicher als Programmspeicher

An den Aufgaben im letzten Abschnitt solltest Du gemerkt haben: Maschinenprogramme aus Makrobefehlen lassen sich leichter verstehen und vor allem viel kompakter hinschreiben als eine Folge von Mikrobefehlen.

6.1 Programme im Arbeitsspeicher



Wir hatten ja bereits als Ziel formuliert, die Programme im Arbeitsspeicher des Rechners speichern zu wollen. Das war übrigens nicht immer so: In den ersten Rechnern wie dem Zuse Z3 waren die Programme z.B. auf Lochstreifen gespeichert. John von Neumann hatte die Idee, nicht nur Zahlen sondern auch Programme in ein und demselben Arbeitsspeicher abzulegen. Man spricht in diesem Zusammenhang von der von-Neumann-Architektur.

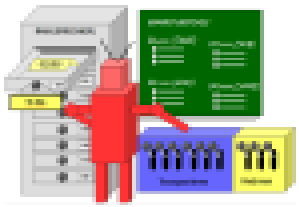
Wie aber lassen sich Makrobefehle im Speicher (RAM) ablegen? Wir sehen, dass der RAM nur Zahlen – in unserem Simulator zwischen 0 und 19999 – speichern kann. Wie also kann man einen Befehl wie

```
TAKE 010
```

im RAM speichern?

Zunächst überlegt man sich hierfür, dass dieser Befehl eigentlich zweigeteilt ist.

- Zum einen ist da die eigentliche Operation mit dem Namen „ADD“. Es wäre naheliegend, zusätzlich zum Namen einfach einen Zahlencode für jede solche Operation festzulegen, den "Operation Code" oder kurz "Op-Code". Wenn wir annehmen, dass wir wahrscheinlich nicht mehr als 15 Befehle brauchen, reichen zwei Dezimalstellen locker, um allen Befehlen eine eindeutige Nummer zu geben. Wir könnten also TAKE z.B. die Op-Code 01 geben (die führende Null, nur um klarzumachen, dass Op-Codes bei uns hier immer zwei Dezialstellen haben).
- Zum anderen ist da auch noch die Adresse, auf die sich der Befehl bezieht. Der Speicher unseres Simulationsrechners enthält 1000 Speicherstellen mit Adressen zwischen 0 und 999. Um die Adressen darzustellen brauchen wir also drei weitere Dezimalstellen.



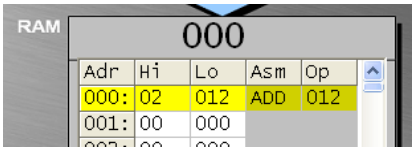
Insgesamt brauchen wir also 5 Stellen, um einen ganzen Befehl wie "TAKE 010" zu speichern. Und -- welch ein Zufall: die Speicherstellen von JOHNNY haben 5 Dezimalstellen.

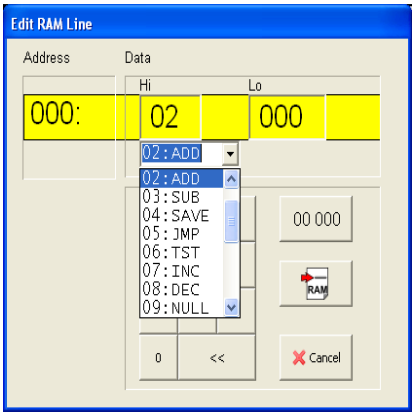
Wir vereinbaren folgendes:

- Die 10000er- und die 1000er-Stelle sollen den Op-Code des Befehls beschreiben. Da die Zahlen im Speicher von JOHNNY Werte von 0 bis 19999, könnte man im Grunde 20 verschiedene Op-Codes definieren, wobei wir Op-Code 00 aus bestimmten Gründen zunächst weglassen, also 19 verschiedene OP-Codes. Mehr als genug für unsere weiteren Überlegungen.
- Die restlichen drei Stellen (100er-, 10er- und 1er-Stelle) sollen die Adresse beschreiben, auf die sich der Befehl bezieht.

Damit wird aus dem Befehl „TAKE 010“ nun einfach die Zahl „01010“. Wir schreiben im folgenden den in der Geschäftswelt üblichen Dezimalpunkt hin, um die beiden Teile besser auseinanderhalten zu können: „01.010“.

6.2 Umsetzung in JOHNNY





Damit man sich als Benutzer des Simulators die Zahlen für die einzelnen Operationen nicht merken muss, wurden im Simulationsprogramm zwei Erleichterungen eingeführt:

- Bei der Darstellung des Arbeitsspeichers werden hinter den eigentlichen Speicherstellen als Kommentar die Makrobefehle angezeigt (falls die Zahl möglicherweise einem Makrobefehl entspricht).
- Bei der Eingabe von Werten in den RAM kann man ein Auswahlmenü anklicken, das direkt die einzelnen Operationen mit ihren entsprechenden Zahlen zur Auswahl stellt.
- Eigentlich ist es egal, an welcher Speicherstelle des RAM ein Programm beginnt. Praktisch ist es allerdings, wenn es an der Speicherstelle 000 beginnt, weil man dann mit der RESET-Taste in der Werkzeugleiste direkt zum ersten Befehl springen kann.

Aufgabe: Programmtest

Schreiben Sie das folgende Programm ab Speicherstelle 000 in den Arbeitsspeicher und testen Sie, ob es tut, was es soll. Welche Tasten in der Symbolleiste welche Funktion haben, können Sie sicherlich leicht herausfinden.

```
000:    TAKE 010
001:    ADD  011
002:    ADD  012
003:    SUB  014
004:    SAVE 015
005:    HLT  000
```

Damit eine Überprüfung leichter wird, ist es sinnvoll, die Speicherzellen auch mit von Null verschiedenen Zahlenwerten zu belegen, etwa:

```
010:    42
011:    23
012:    137
013:    0
014:    99
015:    25
```

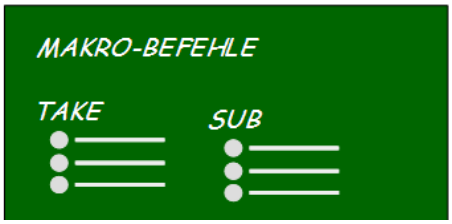
Beschreiben Sie, was der Befehl HLT am Ende tut!

(Hinweis: die Adresse hinter dem HLT-Befehl ist vollkommen uninteressant).

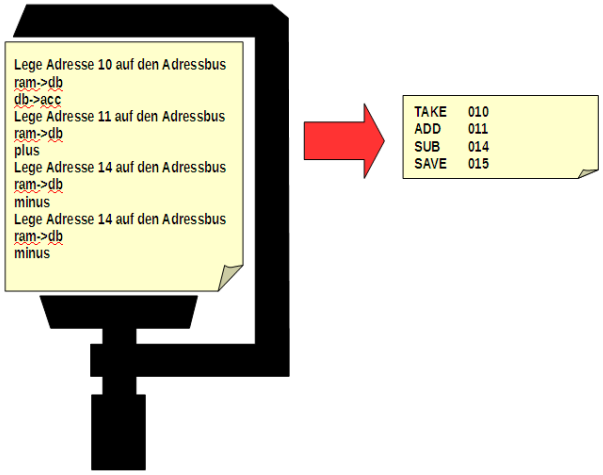
Der Speicher als Programmspeicher -- Lösung A.1

Der Halt-Befehl zeigt mit einem Fenster im Simulator an, dass das Programm anhält. Lässt man das Programm automatisch durchlaufen (Taste mit Doppelpfeil: Run Macro Code), so stoppt der Befehl HLT die Automatik.

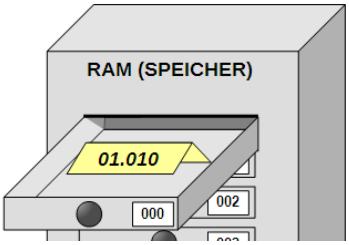
6.3 Fachkonzept: Makrobefehle

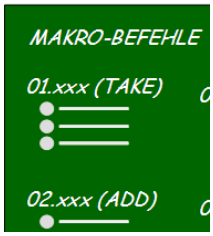


- Makrobefehle fassen jeweils eine Reihe von Mikrobefehlen zusammen.

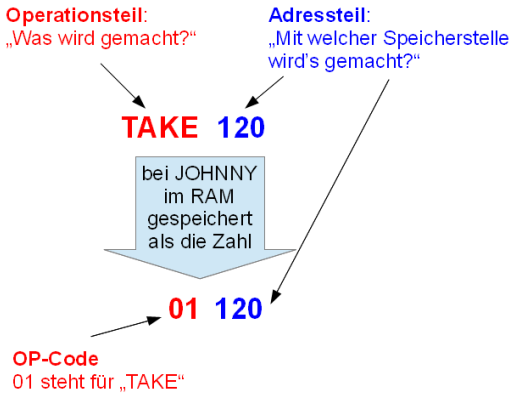


- Programme aus Makrobefehlen sind übersichtlicher und kompakter als Folgen von Mikrobefehlen. Das ist gut für Programmierer und auch für den "Speicherverbrauch".





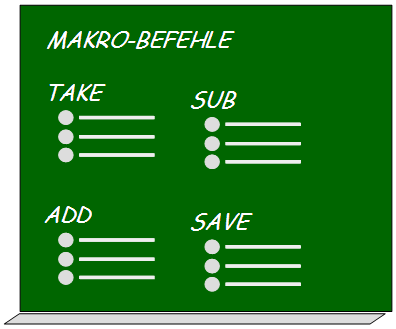
- Die einzelnen Makrobefehle inklusive der Adressen, auf die sie wirken werden als Daten im Arbeitsspeicher abgelegt. Letztlich sind solche Programmdaten nicht von anderen Daten unterscheidbar: sie können als Daten gelesen, geschrieben und verändert werden. Eben diese Idee geht auf John von Neumann zurück, nach dem (aus diesem Grund) der JOHNNY-Simulator benannt ist: Man spricht von der "von-Neumann-Architektur".



- Der Befehl selbst (der "Operationsteil) wird dabei als Zahl (Op-Code) beschrieben, die Adresse ist ja bereits eine Zahl. Bei JOHNNY teilen sich OP-Code und Adresse nach dem Schema OP.ADR eine Speicherzelle. Das muss aber nicht bei jedem Computer so sein; ein Befehl kann sich auch auf mehrere hintereinander liegende Speicherstellen verteilen.

Wie der Computer aus einem Befehl in OP.ADR-Schreibweise formulierten Befehl tatsächlich die Folge von Mikrobefehlen ermittelt und ausführt, wird später in Abschnitt 5: "Das Steuerwerk" thematisiert. Zunächst einmal werden wir uns jetzt in Abschnitt 4: "Programmieren mit JOHNNY" anschauen, was man mit den in JOHNNY eingebauten Makrobefehlen alles anstellen kann.

7 Weitere Befehle bei JOHNNY



Reichen die bisher betrachteten Befehle "TAKE", "ADD", "SUB", "SAVE" und "HLT" schon aus, um alle möglichen Programme zu schreiben und das auch noch komfortabel.

Aufgabe 1: Test auf Brauchbarkeit

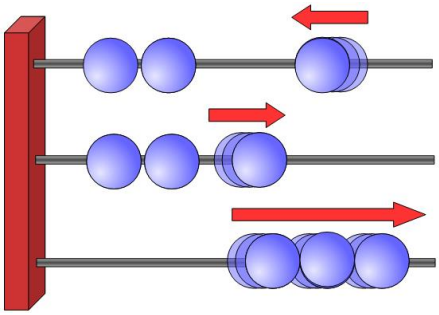
Testen wir einfach mal an ein paar Problemen, ob die fünf genannten Befehle zum komfortablen Programmieren ausreichend sind. Bewerten Sie zunächst jedes der Probleme, ob es mit den fünf Befehlen lösbar ist. Falls Sie zu dem Schluss kommen, es sei lösbar, programmieren Sie es in JOHNNY.

- **Erhöhung einer Zahl um 1**
  Der Computer soll die Zahl in Speicherzelle 10 um 1 erhöhen.
- **Countdown von 10 bis 0**
  Der Computer soll in der Speicherzelle 12 zunächst die Zahl 10 stehen haben und dann schrittweise bis 0 herunterzählen.
- **Countdown von einer beliebigen Zahl bis 0**
  Der Computer soll in der Speicherzelle 12 zunächst irgendeine Zahl 10 stehen haben und diese dann schrittweise bis 0 herunterzählen.

Aufgabe 2: Wunschkonzert

Formulieren Sie, welche zusätzlichen Möglichkeiten Sie sich von JOHNNY wünschen (würden).

7.1 Ein paar praktische Befehle



Wahrscheinlich haben Sie bereits festgestellt, dass es praktisch wäre, noch ein paar besondere Befehle beim Programmieren direkt zur Verfügung zu haben. Statt hier einfach zu beschreiben, welche Befehle JOHNNY noch auf Lager hat, können Sie es durch Ausprobieren auch selbst herausfinden.

Aufgabe 1: Wirkungsweise von Befehlen erschliessen

Speicheren Sie zunächst in den Speicherstellen 10 und 11 jeweils die Zahl 42. Geben Sie das folgende Programm in JOHNNY ein und beobachten Sie, wie sich bei jedem Befehl die Speicherstellen 10 und 11 verändern.

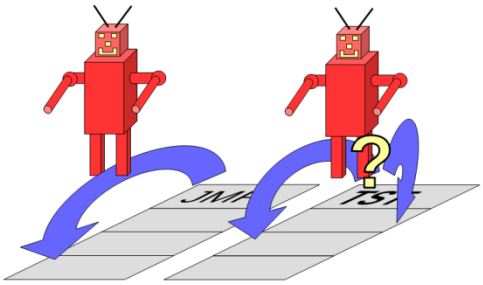
```
000:    INC  010
001:    INC  010
002:    INC  011
003:    DEC  010
004:    INC  010
005:    INC  010
006:    NULL 010
007:    NULL 011
008:    HLT  000
```

Fertigen Sie eine Tabelle an, in der Sie die Wirkung der Befehle "INC", "DEC" und "NULL" sehr exakt formulieren.

Aufgabe 2: Reverse Engineering von Mikrobefehlsfolgen

Überlegen Sie sich die Mikrobefehlsfolgen für die drei Befehle "INC", "DEC" und "NULL".

7.2 Sprünge, Verzweigungen und Schleifen



Bisher liefen unsere JOHNNY-Programme bei Speicherstelle 0 los und schrittweise durch das Programm. Solange, bis sie schliesslich beim HLT-Befehl angekommen waren. Wir wir schon gesehen haben, reicht ein solcher gerader Durchlauf von vorne nach hinten zur Lösung mancher Probleme aber nicht aus.

Aufgabe 1

Testen Sie die beiden folgenden Programm und notieren Sie sich danach, was der Befehl "JMP" tut.

Programm 1:

```
000:    INC   010
001:    JMP   004
002:    INC   011
003:    INC   011
004:    INC   012
005:    HLT   000
```

Programm 2:

```
000:    INC   010
001:    INC   011
002:    INC   012
003:    JMP   001
004:    HLT   000
```

Überlegen Sie, wie sinnvoll in diesem Programm eigentlich der HLT-Befehl beim Programm 2 ist.

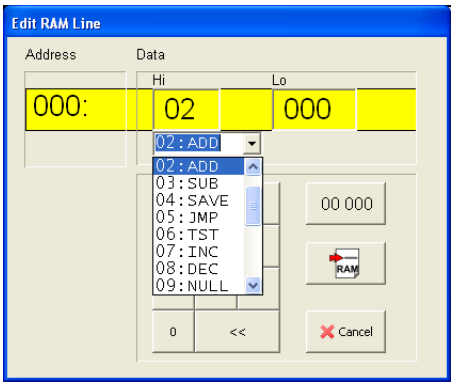
Aufgabe 2

Testen Sie das folgende Programm und beobachten Sie dabei die Werte der Speicherstelle 10. Notieren Sie sich, was der Befehl "TST" tut.
Hinweis: Der Befehl "TST" tut nicht immer etwas.

```
000:    INC   010
001:    INC   010
002:    INC   010
003:    INC   010
004:    DEC   010
005:    TST   010
006:    JMP   004
007:    INC   010
008:    HLT   000
```

Es kann sein, dass Sie mehrere Programmläufe brauchen, um die Lösung für diese Aufgabe zu sehen: "TST" ist der schwierigste Befehl im JOHNNY-Assembler.

7.3 Fachkonzept: (JOHNNY-)Assembler



Assembler-Befehle bei JOHNNY

Fassen wir die JOHNNY-Assembler-Befehle noch einmal systematisch zusammen. Wir können dabei drei Gruppen von Befehlen ausmachen:

- Befehle, die nur Daten transportieren (Transportbefehle)
- Befehle, die (auch) Rechenoperationen ausführen (Rechenbefehle)
- Befehle, die den Ablauf des Programms steuern (Programmablauf-Befehle)

Transportbefehle

Assembler-Befehl	Funktion
`TAKE` Adresse	Transportiert die Zahl aus der angegebenen Adresse in den Akkumulator des Rechenwerks
`SAVE` Adresse	Transportiert die Zahl aus dem Akkumulators des Rechenwerks zur angegebenen Adresse

Rechenbefehle

Assembler-Befehl	Funktion
`ADD` Adresse	Addiert die Zahl aus der angegebenen Adresse zum Akkumulator des Rechenwerks Besonders bei JOHNNY: Wäre das Ergebnis grösser als 19999, so wird es auf 19999 gesetzt.
`SUB` Adresse	Subtrahiert die Zahl aus der angegebenen Adresse vom Akkumulator des Rechenwerks Besonders bei JOHNNY: Wäre das Ergebnis kleiner als 0, so wird es auf 0 gesetzt.
`INC` Adresse	Erhöht die Zahl an der entsprechenden Adresse um 1 Besonders bei JOHNNY: Wäre das Ergebnis grösser als 19999, so wird es auf 19999 gesetzt.
`DEC` Adresse	Erniedrigt die Zahl an der entsprechenden Adresse um 1 Besonders bei JOHNNY: Wäre das Ergebnis kleiner als 0, so wird es auf 0 gesetzt.
`NULL` Adresse	Setzt die Zahl an der entsprechenden Adresse auf den Wert 0

Programmablauf-Befehle

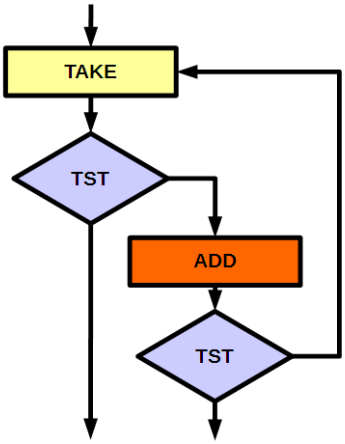
Assembler-Befehl	Funktion
`JMP` Adresse	Setzt das Programm an der angegebenen Adresse fort
`TST` Adresse	Testet die Zahl bei der angegebenen Adresse. Ist die Zahl nicht 0, tut der Befehl selbst fast gar nichts, sondern es wird einfach zum nächsten Befehl weitergegangen. Ist die Zahl hingegen 0, wird im weiteren Verlauf des Programms der nächste (also der auf den TST-Befehl folgende) Befehl übersprungen.
`HLT` Adresse	Das Programm hält definiert an. Die angegebene Adresse ist vollkommen unerheblich.

Allgemeines zu Prozessoren und Assembler-Sprachen

Reale Computer haben einen im Grunde ähnlichen Befehlssatz, allerdings meistens deutlich mehr und auch kompliziertere Befehle. Dabei unterscheidet man zwei verschiedene "Philosophien".

- Manche Prozessoren haben dabei sehr viele Befehle. Man spricht auch von Computern mit einem komplexen Befehlssatz (engl: Complex Instruction Set Computer, CISC). Ein Beispiel ist der Befehlssatz für Intel x86-kompatible Computer, wie die meisten PCs: Intel-x86-Befehlssatz.
- Bei anderen Prozessoren versucht man, die Anzahl der Befehle gering und die Befehle selbst einfach zu halten. Man spricht auch von Computern mit einem reduzierten Befehlssatz (engl: Reduced Instruction Set Computer, RISC).

8 Programmieren mit JOHNNY



Kann man mit JOHNNY und seinen Befehlen Dinge tun, die man mit echten Computern auch kann? Gut: Graphik-Programmierung wird wohl ein bisschen schwierig, denn JOHNNY hat keinen Bildschirm. Also begnügen wir uns mit Standard-Problemen, wie sie auch bei den meisten Computerprogrammen auftauchen:

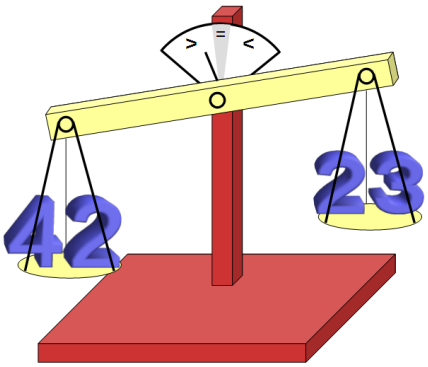
- **Vergleiche**
  Vergleiche von Zahlen spielen beim Programmieren eine wichtige Rolle, nicht nur bei offenkundig mathematischen Programmen, sondern auch und gerade bei Computerspielen.
- **Kompliziertere Rechenoperationen**
  JOHNNY hat eingebaut nur Befehle zum Addieren und Subtrahieren. Das ist für viele Probleme aus der wirklichen Welt natürlich eindeutig zu wenig.

Neben diesen Dingen, die man aus praktisch allen Programmiersprachen kennt, können Programme in JOHNNY dank der von-Neumann-Architektur (Programme liegen als Daten im Speicher vor), etwas, was viele andere Programmiersprachen nicht (so ohne weiteres) können: sie können sich selbst ändern. Das hat Vor-, aber auch Nachteile:

- **Vorteile**
  Der Computer kann auf diese Weise sozusagen selbst Maschinen-Programme schreiben. Das muss allerdings selbst wieder ein Programm erledigen, was ist also der Sinn der Sache? Sinnvoll wird die Sache, wenn das Maschinen-Programme-schreibende Programm selbst auf Programmtexte in einer anderen Programmiersprache wie C, C++ oder Pascal zurückgreift und sie in Folgen von Maschinenbefehlen übersetzt. Solche Übersetzungsprogramme nennt man Compiler.
- **Nachteile**
  Was für die Konstruktion von Compilern sehr hilfreich ist, macht Computersysteme aber auch angreifbar. Bösartige Software wie Computerviren kann dann nämlich Befehlsfolgen anderer Programme im Speicher ändern. Dies kann zur Zerstörung und zum Ausspionieren von Daten, aber auch zur Vermehrung der Viren selbst eingesetzt werden.

8.1 Vergleiche

Vergleiche von Zahlen sind in der Mathematik, aber natürlich auch im praktischen Leben, eine ganz wichtige Sache. Wenn man einkauft, könnte es wichtig sein zu wissen, ob der Preis höher ist als das zur Verfügung stehende Geld.



Aufgabe 1: Einfacher Vergleich: "Echt grösser"

Wir nehmen an, in den Speicherstellen mit Adressen 10 und 11 liegen zwei Zahlen. Schreiben Sie ein Programm, dass in Speicherzelle 15 eine 1 schreibt, wenn die Zahl bei Adresse 10 echt grösser ist als die bei Adresse 11, sonst soll in Speicherzelle 15 eine 0 stehen.

Wir schreiben die Bedingung als [10] > [11], wobei die eckigen Klammern um die Zahlen andeuten sollen, dass hier nicht die Zahlen 10 und 11 gemeint sind, sondern die Zahlen, die an den entsprechenden Adressen gespeichert sind.

Aufgabe 2: Ein etwas aufwändigerer Vergleich: "Grösser oder gleich"

Ändern Sie das Programm aus Aufgabe 1 so ab, dass in Speicherzelle 15 die 1 bereits dann steht, wenn die beiden Zahlen gleich gross sind. Also Bedingung: [10]>=[11]

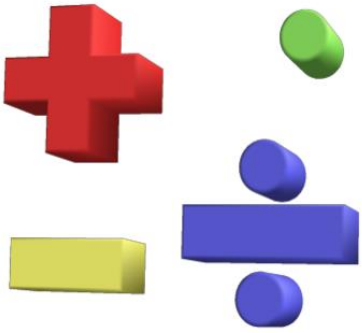
Was so einfach klingt, ist bei JOHNNY deshalb nicht einfach, weil es keinen Unterschied macht, ob man zwei gleich grosse Zahlen voneinander subtrahiert (Ergebnis: 0) oder eine grössere von einer kleineren Zahl (Ergebnis: ebenfalls 0). Da wir es jedoch nur mit ganzen Zahlen zu tun haben, der kleinste Abstand zwischen zwei verschiedenen Zahlen also 1 ist, gibt es einen einfachen Trick, um das Problem trotzdem zu lösen.

Aufgabe 3: Luxus-Vergleich

Wir wollen nun die beiden Aufgaben quasi kombinieren. Das Ergebnis in Speicherzelle 15 soll folgende Werte annehmen:

- 1 : falls [10]>[11]
- 2 : falls [10]=[11]
- 3 : falls [10]<[11]

8.2 Mehr Rechenarten



Bestimmt hast Du Dich schon gefragt, weshalb JOHNNY nur die Rechenarten Addition und Subtraktion beherrscht und nicht etwa auch die Multiplikation und die Division. Schliesslich lernt man das ja schon in der Grundschule und welcher Rechner könnte sich schon "Rechner" nennen, wenn er die beiden letzteren nicht auch beherrschen würde. JOHNNY beherrscht das nicht, jedenfalls nicht als eigener Befehl. Aber trotzdem kann man JOHNNY dazu bringen, zwei Zahlen im Speicher zu multiplizieren und das Ergebnis abzuspeichern.

Aufgabe 1: Multiplikation mit festem Faktor

Schreiben Sie ein Programm, das die Inhalte der Speicherstellen 10 mit der Zahl 4 multipliziert. Das Ergebnis soll nachher in der Speicherstelle Nummer 12 gespeichert sein:
[12]:=[10] * 4

Aufgabe 2: Multiplikation mit variablem Faktor

Schreiben Sie ein Programm, das die Inhalte der Speicherstellen 10 mit dem Inhalt der Speicherstelle 11 multipliziert. Das Ergebnis soll nachher in der Speicherstelle Nummer 12 gespeichert sein:
[12]:=[10] * [11]

Tipp: Der Inhalt der Speicherstelle 11 darf am Ende der Berechnung ruhig den Wert 0 haben.

Aufgabe 3: Test eines Spezialfalls

Testen Sie das Programm aus Aufgabe 2 für die beiden Fälle, dass [10]=0 bzw. dass [11]=0 ist. Falls das Programm für diesen Fall nicht korrekt funktioniert, änderen Sie es entsprechend ab.

Aufgabe 4: Division teilbarer Zahlen

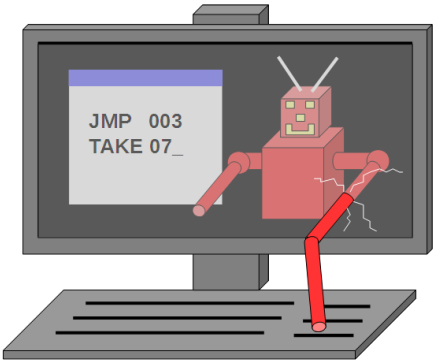
Wir nehmen an, in den Speicherstellen 10 und 11 stehen zwei Zahlen, wobei [10] durch [11] teilbar ist. Schreiben Sie ein Programm, das den Inhalte der Speicherstellen 10 durch denjenigen in Speicherzelle 11 dividiert. Das Ergebnis soll nachher in der Speicherstelle Nummer 12 gespeichert sein:
[12]:=[10] / [11]

Aufgabe 5: Ganzzahlige Division mit Rest

In der Grundschule haben wir gelernt wie man Zahlen mit Rest dividiert:
"In die Zahl 17 passt die Zahl 3 genau 5 mal hinein und es bleibt ein Rest von 2."
Wir nehmen an, in den Speicherstellen 10 und 11 stehen zwei beliebige Zahlen. Schreiben Sie ein Programm, das den Inhalte der Speicherstellen 10 durch denjenigen in Speicherzelle 11 ganzzahlig mit Rest dividiert. Das Ergebnis soll nachher in der Speicherstelle Nummer 12 gespeichert sein:

[12]:=[10] // [11] ("//": Symbol für ganzzahlige Division)
[13]:=[10] % [11] ("%": Symbol für Rest bei der ganzzahligen Division)

8.3 Programme ändern sich selbst



In einem der vorhergehenden Abschnitte haben wir gesagt, dass John von Neumann die Idee hatte, die Programme im Arbeitsspeicher abzulegen, statt direkt von einem externen Datenträger auszuführen, wie das etwa beim deutschen Rechner ZUSE Z3 noch der Fall war. Unabhängig davon, dass der Arbeitsspeicher eines Rechners meist sehr schnell ist, ergeben sich daraus noch ganz andere Möglichkeiten.

Aufgabe 1: Programm-Analyse

Überlegen Sie zunächst theoretisch, was das folgende Programm tut und notieren Sie in einer kurzen Darstellung Ihre Überlegungen. Testen Sie dann am Simulator, ob die Ergebnisse des Simulationslauf zu Ihren Überlegungen passen.

Beobachten Sie vor allem, was im Laufe der Ausführung mit der Speicherstelle 001 geschieht.

```
000:        TAKE 007
001:        SAVE 100
002:        INC  001
003:        DEC  008
004:        TST  008
005:        JMP  000
006:        HLT  000
007:             042
008:             013
```

Aufgabe 2: Reihenweise Zahlen

a)
Änderen Sie das Programm so ab, dass es nicht immer die gleiche Zahl in den Speicher schreibt, sondern fortlaufende Zahlen, also etwa 42, 43, 44, usw.

b)
Führen Sie eine weitere Änderung durch, und zwar so, dass die Zahlen nur in die geraden Speicherstellen ab Speicherstelle 100 geschrieben werden, also 100, 102, 104, usw.

Aufgabe 3: "Computervirus"

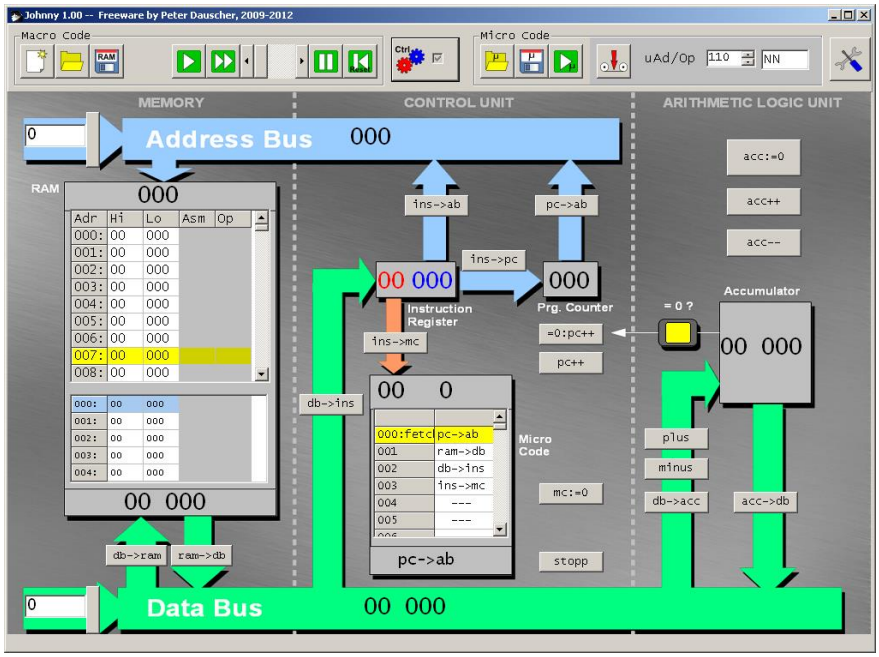
a)
Schreiben Sie ein Programm, das sich selbst im Speicher vermehrt, also sich selbst an eine andere Stelle des Speichers, etwa ab Speicherstelle 50 kopiert.

b)
Jetzt wird's echt schwierig: Änderen Sie das Programm so ab, dass es nach dem "Vermehrungsvorgang" in die Kopie spingt, und diese sich wiederum vermehrt.

9 Das Steuerwerk

Bisher haben wir einfach akzeptiert, dass unser JOHNNY-Rechner die einzelnen Befehle im Arbeitsspeicher von 000 an ausführt, ohne zu fragen, wie er das eigentlich macht.

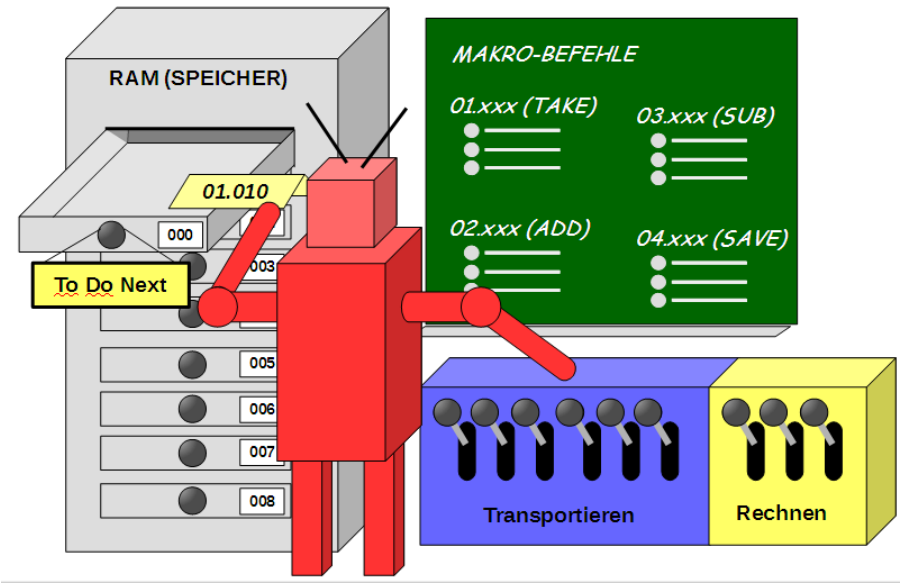
Hierfür ist das Steuerwerk (englisch: Control Unit) verantwortlich. Diese hatten wir die ganze Zeit der Übersichtlichkeit halber verdeckt gelassen. Jetzt aber können wir mutig sein und mit einem Druck auf die Taste "Ctrl" (die Taste mit den Zahnrädern) das Steuerwerk einblenden.



Man erkennt, dass das Steuerwerk aus einer Reihe von miteinander verbundenen Teilen besteht, und dass es auch acht neue Tasten gibt, also acht zusätzliche Mikrobefehle.

9.1 Aufbau und Funktion des Steuerwerks

Nachdem wir nun gesehen haben, dass man mit den Makro-Befehlen richtig komplizierte Programme schreiben kann, kehren wir nochmal zurück zu der Frage, wie eigentlich die Makrobefehle genau in die Folgen von entsprechenden Mikrobefehlen übersetzt werden.



Versetzen wir uns in die Lage des kleinen roten Roboters. Er symbolisiert in dieser Graphik das Steuerwerk.
Nehmen wir an, dieser Roboter kommt an seinen "Arbeitsplatz" und findet, ordentlich geschlossen, den RAM-Schrank vor. An einer Schublade baumelt das Schild "To Do Next". Er wird nacheinander folgende Schritte tun:

- Schublade mit dem Schild "To Do Next" suchen.
- Diese Schublade öffnen.
- Darin liegenden Zettel in die linke Hand nehmen.
- Die linken zwei Stellen, in diesem Fall die 01, auf der Tafel der Makrobefehle suchen. und beginnen, die entsprechende Folge von Mikrobefehlen abzuarbeiten.

Aufgabe 1: Experiment mit dem JOHNNY-Steuerwerk

Um herauszufinden, ob die einzelnen Teile des Steuerwerk auch so ähnlich funktionieren, legen Sie zunächst die Zahl 01.010 in die Speicherstelle 000. Schaut man nach, dann sieht man, dass dies genau dem Befehl

```
TAKE 010
```

entspricht. Dann drücke nacheinander auf die Tasten für die folgenden Mikrobefehle (die Play-Taste mit dem kleinen griechischen µ).
Notieren Sie die Folge an Mikro-Befehlen, die nach den einzelnen Tastendrücken ausgeführt werden.

Aufgabe 2: Vergleich: Karikatur und JOHNNY-Steuerwerk

Versuchen Sie, auf der Oberfläche von JOHNNY diejenigen Teile des Steuerwerks zuzuordnen, die den folgenden Teilen der Karikatur entsprechen:

- Der Zettel in der Hand des Roboters
- Das Schild "To Do Next"
- Die Tafel mit den Mikrobefehls-Listen für die einzelnen Makrobefehle.

Vesuchen Sie ausserdem, den folgenden Mikrobefehlen die entsprechende Handlungen unseres Roboters zuzuordnen.

ins->ab

ins->pc

pc++

Aufgabe 2: Tabelle aller Mikrobefehle des Steuerwerks

Schreiben Sie folgendes Programm in den Arbeitsspeicher:

```
000:        TAKE 010
001:        DEC  010
002:        TST  010
003:        JMP  001
```

004: HLT

Arbeiten Sie es schrittchenweise mit der "Play-µ-Taste" ab und beobachten Sie dabei die Wirkungsweise der nacheinander ausgeführen Mikrobefehle. Dabei sollten Ihnen auch die Mikrobefehle "`stop`", "`mc:=` und "`=0:pc++` begegnen. Erstellen Sie nun (auf Papier oder in einer Datei auf dem Rechner) eine Tabelle mit allen Mikrobefehlen des Steuerwerks und einer Beschreibung ihrer Funktion.

9.2 Der von-Neumann-Zyklus

Aufgabe 1: Abschluss von Makrobefehlen

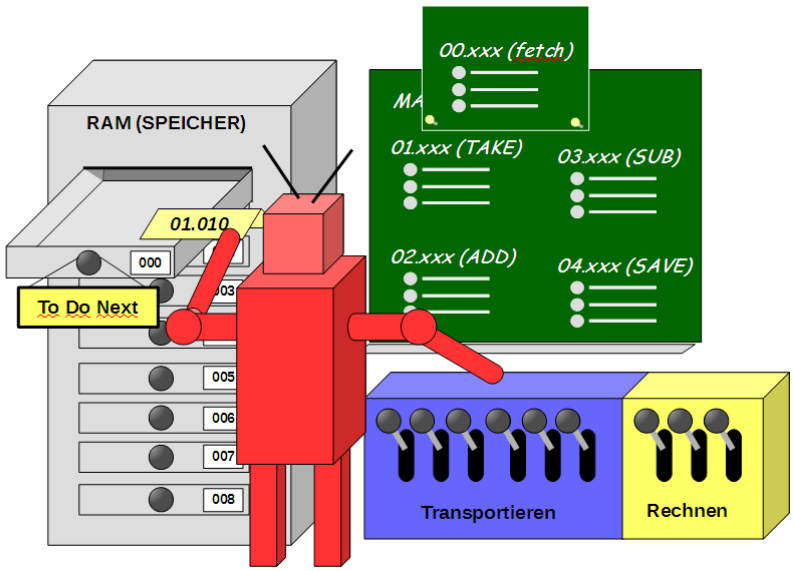
Die Mikrobefehls-Folgen aller Makrobefehle (mit Ausnahme von `JMP`) enden auf

```
...
pc++
mc:=0
```

Erläutern Sie, weshalb das so ist und weshalb `JMP` eine Ausnahme ist.

Aufgabe 2: Nochmal die Karikatur

Wahrscheinlich ist Ihnen bereits aufgefallen, dass die Folge von Mikrobefehlen, die notwendig sind, um die Befehle vom RAM ins Instruction Register (deutsch: Befehlsregister) zu transportieren, noch gar nicht in unserer Karikatur vorkommen. Das wollen wir zum Abschluss dieses Abschnitts noch schnell ändern.



Beschreiben Sie, welche Handlung des Roboters anschaulich dem Mikrobefehl `mc:=0` entspricht.

Aufgabe 3: JMP und TST im Micro-Code

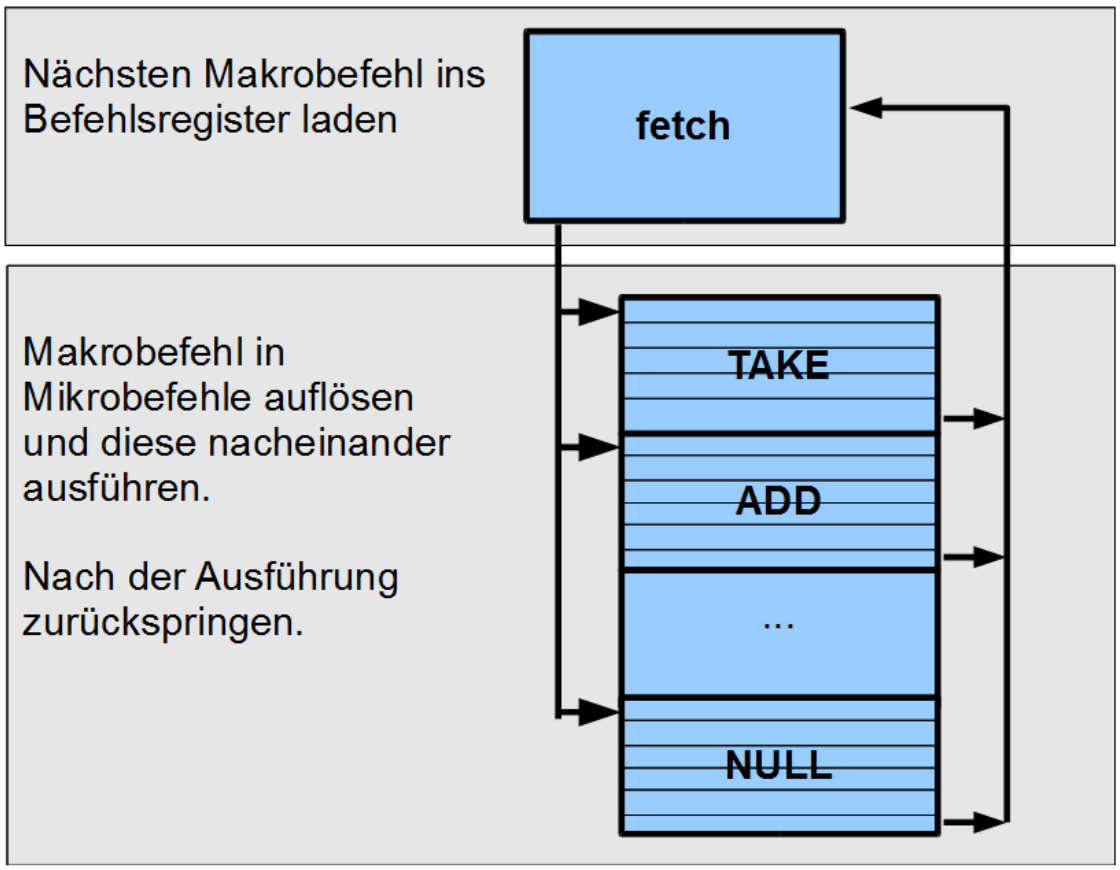
Überlegen Sie zunächst selbst, wie der Inhalt des Mikrobefehlsspeicher für die Befehle "JMP" und "TST" aussieht. Notieren Sie zunächst Ihre Vermutungen. Vergleichen Sie dann mit dem Inhalt des Mikrobefehlspeichers bei JOHNNY.

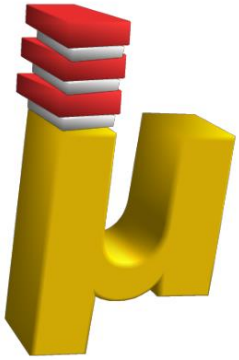
9.3 Fachkonzept: von-Neumann-Zyklus

Damit ein Programm aus Makrobefehlen im Arbeitsspeicher abgearbeitet werden kann, werden in einem ständigen Wechsel immer zwei Schritte durchgeführt:

1 Holen des jeweiligen Befehls aus dem Arbeitsspeicher ins Befehlsregister (Instruction Register). Welcher Befehl gerade geholt werden soll, ist im Programmzähler (Program Counter) vermerkt.

2 Befindet sich der Befehl im Befehlsregister, dann wird im Mikrobefehlszähler an die entsprechende Stelle gesprungen und die zugehörige Folge von Mikrobefehlen ausgeführt. Danach wird dann wieder zum Schritt 1 gesprungen und der nächste Befehl ins Befehlsregister geladen.





9.4 Eigene Mikrobefehle

Bei Mikroprozessoren ist die Menge an Makrobefehlen und ihre Umsetzung im Steuerwerk fest vorgegeben. Man speicht in diesem Zusammenhang auch vom Befehlssatz eines Prozessors. Bei JOHNNY kann man (ähnlich wie bei frühen Grossrechnern) allerdings eigene Befehle konstruieren und schauen, ob und wie sie funktionieren.

Aufgabe 1: Ergänzen des Micro-Codes

Finden Sie anhand der JOHNNY-Dokumentation (normalerweise beim Programm dabei) heraus, wie man Folgen von Mikrobefehlen zu einem eigenen Makrobefehl mit selbst gewählten Namen zusammenstellen oder einen bereits bestehenden Befehl in seiner Mikrobefehlsfolge abändern kann.

Aufgabe 2: Optimierung

Betrachten Sie die Folge von Mikrobefehlen zum Makrobefehl TAKE. Sie werden feststellen, dass man diese Folge noch optimieren kann, so dass man einen Schritt bei der Abarbeitung spart. Ändern Sie den Micro-Code entsprechend.

Aufgabe 3: Verdoppeln als eigener Befehl

Angenommen, das Verdoppeln von Zahlen wäre eine sehr sehr wichtige Rechenoperation. Konstruieren Sie einen eigenen Befehl mit Namen DBL zum Verdoppeln einer Zahl an der angegebenen Adresse.