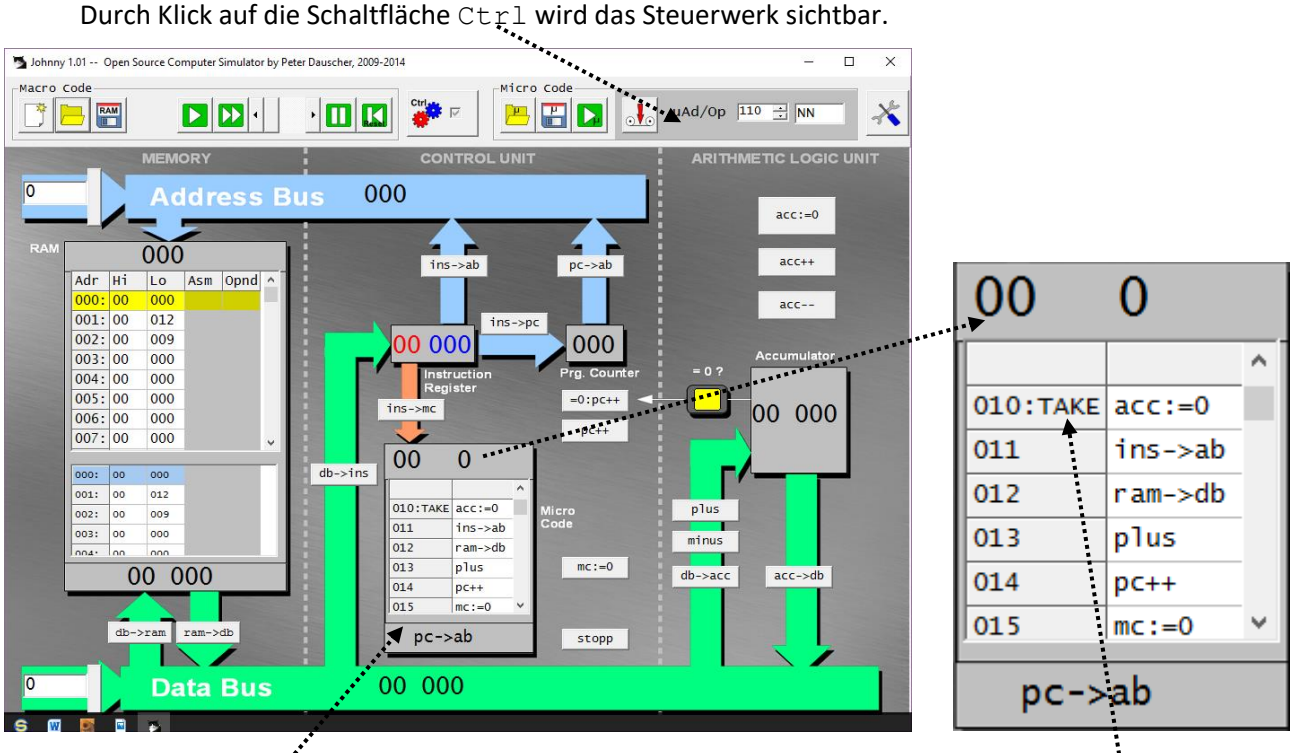
Funktionsweise eines von-Neumann-Rechners II

Bislang hast du gesteuert, in welcher Reihenfolge was mit den Daten zu geschehen hat. Ein Rechner muss sich aber selbst steuern können. Das geschieht in dem **Steuerwerk** (Control Unit). Voraussetzung dafür ist, dass – und das ist der Kern der von-Neumann-Architektur – **der Speicher Daten und Befehle enthält**.

1. Starte Johnny und öffne den gespeicherten RAM aus dem Arbeitsblatt 05, S. 3

   Durch Klick auf die Schaltfläche `Ctrl` wird das Steuerwerk sichtbar.



Jetzt siehst du den **Mikrobefehlsspeicher** mit Anweisungen in der rechten Spalte, die du zuvor durch Mausklicks ausgeführt hast. Das Steuerwerk kann aber auch selbstständig Daten über das Bussystem hin und her bugsieren. Solche Anweisungen nennt man *Mikrobefehle*.

Die Mikrobefehle werden zu einem *Makrobefehl* in der linken Spalte zusammengefasst.

2. Blättere im Mikrobefehlsspeicher mit der Bildlaufleiste nach unten, um dir den Makrobefehl `TAKE` anzeigen zu lassen. Du siehst, der Befehl `TAKE` wird mit `01(0)` codiert.
   Die Null rechts ist die Position in der Mikrobefehlsfolge.

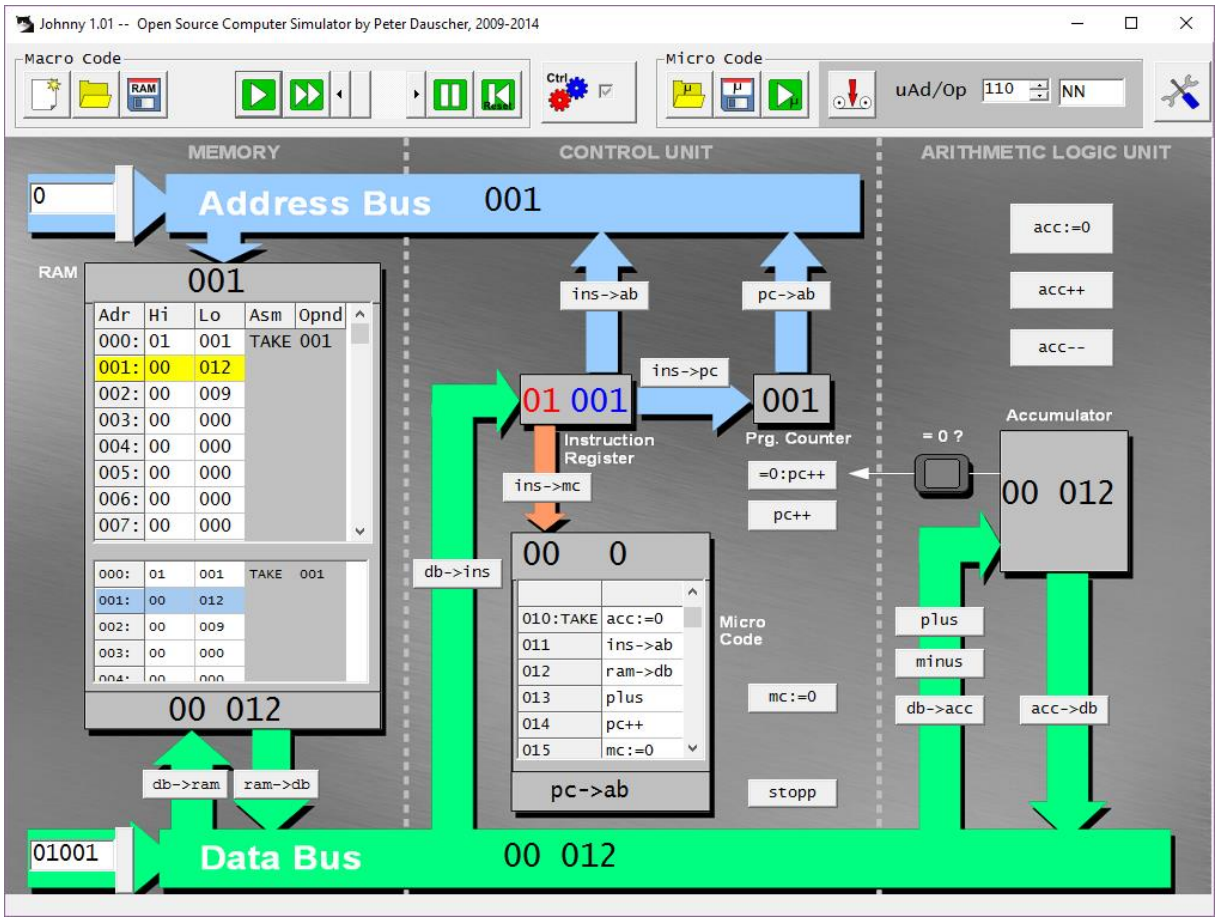
Jetzt wird klar, warum der Speicher in **Hi** und **Lo** unterteilt ist:

- In die Spalte *Hi* werden Makrobefehle eingetragen.
- In die Spalte *Lo* wird die Adresse der Speicherzelle eingetragen, auf die der Makrobefehl zugreift.

3. Trage in der Speicherzelle 000 die Zahl 01001 ein.
   - 000 liegt schon auf dem Adressbus.
   - 01001 auf den Datenbus legen, `db->ram` klicken.

   Jetzt befindet sich im Speicher der Befehl `01`, der auf die Speicherzelle mit der Adresse 001 zugreift.
   Hinweise (*Beobachte die hier beschriebenen Fakten in der Grafik auf der nächsten Seite*):
   - Der Verarbeitungsteil des Rechners (Steuerwerk und Rechenwerk) ist der **Prozessor**.
   - Die Programmiersprache, mit der Prozessoren programmiert werden, nennt man **Assembler**.
   - In der Spalte **Asm** des Speichers (für *Assemblerbefehl*) sieht man die Anweisung im Klartext.
   - Die Speicheradresse, auf die der Befehl zugreift, sieht man in der Spalte **Opnd** (*Operand*).

4. Klicke in Johnny auf die Schaltfläche , um den Befehl auszuführen. Das Ergebnis sieht so aus:



5. Beschreibe die Mikrobefehle, die Johnny ausgeführt hat.

Hinweise:
- Suche die Schaltflächen in Johnny und probiere aus, was sie bewirken.
- Im **Programmzähler** (*Prg. Counter*) wird die RAM-Adresse des aktuellen Makrobefehls mitgezählt.
- Das **Befehlsregister** (*Instruction Register*) enthält den aktuellen Makrobefehl mit dem Operanden.

`acc:=0:`	
`ins->ab:`	
`ram->db:`	
`plus:`	
`pc++:`	
`mc:=0:`	

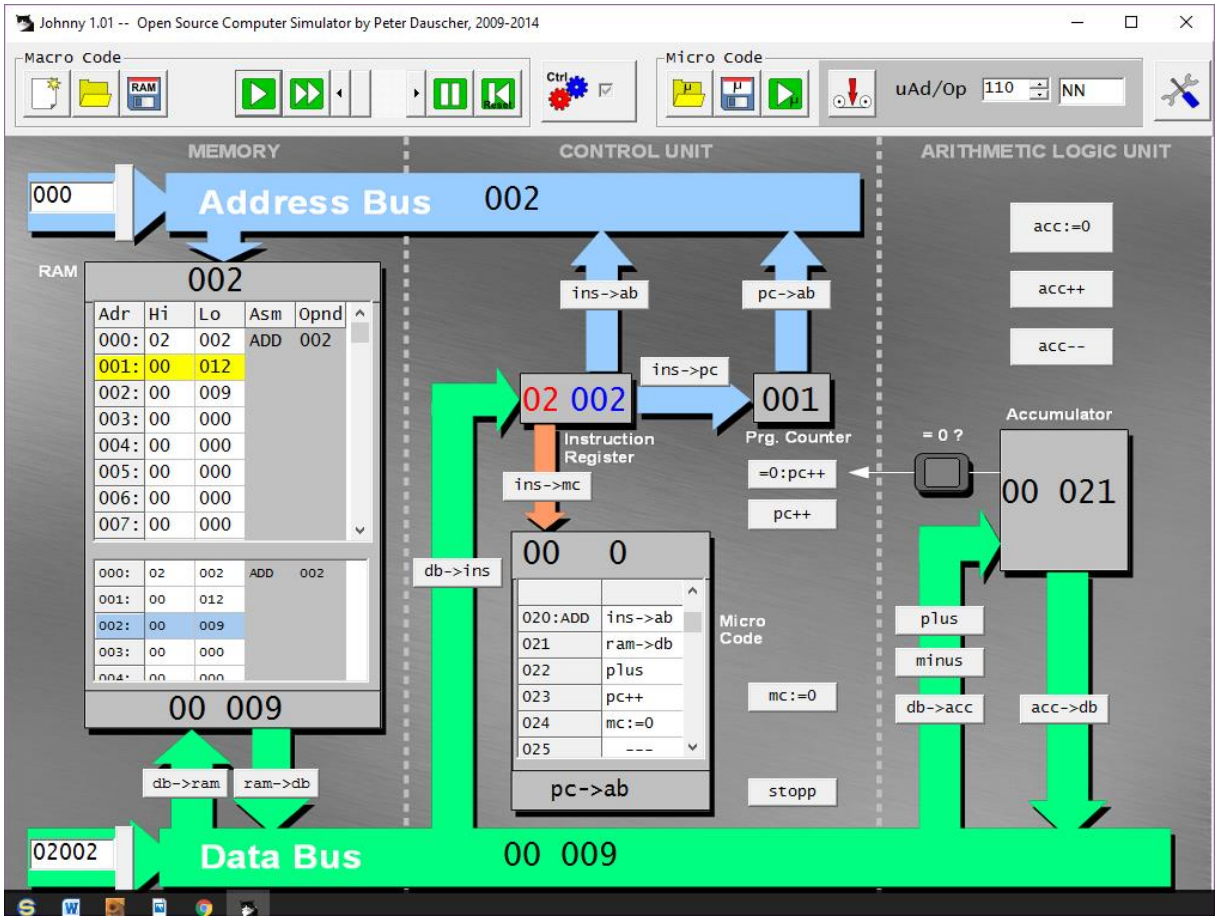
6. Erprobe in Johnny die Schritte, die erforderlich sind, um mit dem vorhandenen RAM die Zahl in der Speicherzelle 002 zum Akkumulator zu addieren.

   Hinweise:
   - Der Makrobefehl für die Addition lautet `ADD` und hat den Code `02`.
   - Zuerst muss das *Befehlsregister* und der *Programmzähler* durch Klick auf `RESET` 

in der Werkzeugleiste gelöscht werden.
   - *Jetzt kann in den Datenbus die 02002 (ADD 002) eingetragen und*

     mit `db->ram` der Befehl in den Speicher kopiert werden.
   - *Zum Abschluss wird der Befehl mit*  *ausgeführt.*

   Das Ergebnis sieht dann so aus:

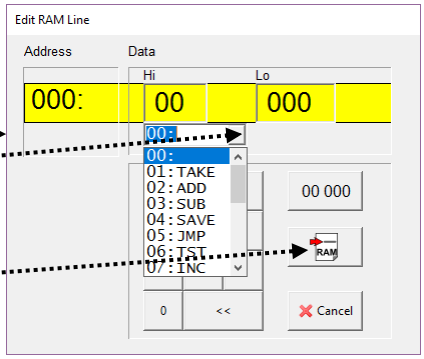


7. Gib den Unterschied zwischen den beiden Makrobefehlen `TAKE` und `ADD` an.

Jetzt hast du Johnny jeweils einen Makrobefehl ausführen lassen. Er kann aber auch eine ganze Folge von Anweisungen abarbeiten, wie du es evtl. auch aus dem Programmierwerkzeug EOS kennst.

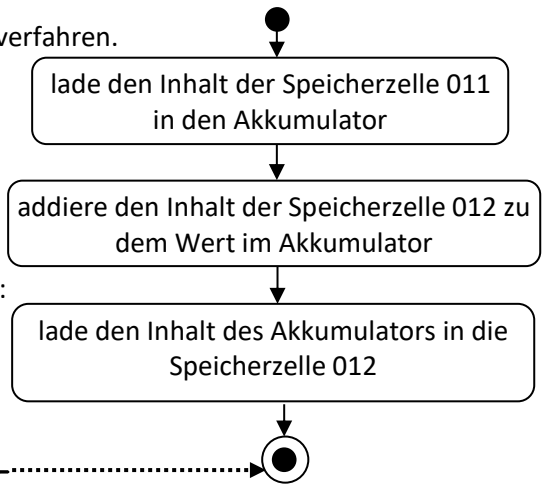
Damit programmierst du dann einen Prozessor in *Assembler*.

8. Beende Johnny und starten ihn neu. Erstelle dann ein Assembler-Programm, das die beiden Zahlen 47 und 11 addiert und das Ergebnis in den Speicher schreibt.
   Hinweise:
   - Nach der Abarbeitung eines Befehls wird der Programmzähler automatisch um eins erhöht. Programme werden also beginnend bei der Speicherzelle 0 aufsteigend abgearbeitet.
     Da zehn Befehle ausreichen werden, sollten die Befehle in die Speicherzellen ab 000 eingetragen werden, die Daten ab 011.
   - Um die Bedienung zu vereinfachen, kann der Speicher auch direkt beschrieben werden, was in Wirklichkeit natürlich nicht möglich ist. Dafür klickst du die betreffende Speicherzelle an.
     Es öffnet sich ein Dialofenster.
     Unter `Hi` kannst du ein Kombinationsfeld aufklicken.
     Wähle den Befehl wird `TAKE` aus, den du schon kennst.
     Unter `Lo` gibst du die Adresse (`011`) ein.
     Mit Klick auf `->RAM` wird der Wert übernommen.
     Jetzt sollte im RAM der folgende Eintrag vorhanden sein:



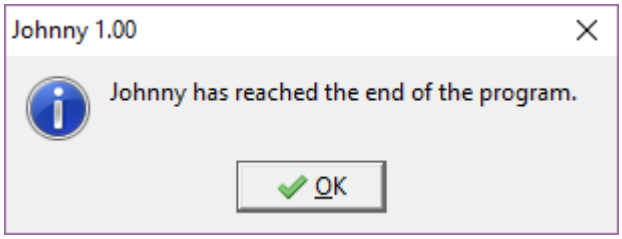
Adr	Hi	Lo	Asm	Opnd
000:	01	011	TAKE	011

   Mit den weiteren Befehlen kannst du dann genauso verfahren.
   Um in den Speicherzellen 011 und 012 einzutragen, wird nur unter `Lo` ein Wert erfasst.
   - Der Programmablauf kann auch hier in einem Aktivitätsdiagramm dargestellt werden, wie du es evtl.schon von der Programmierung mit EOS her kennst.



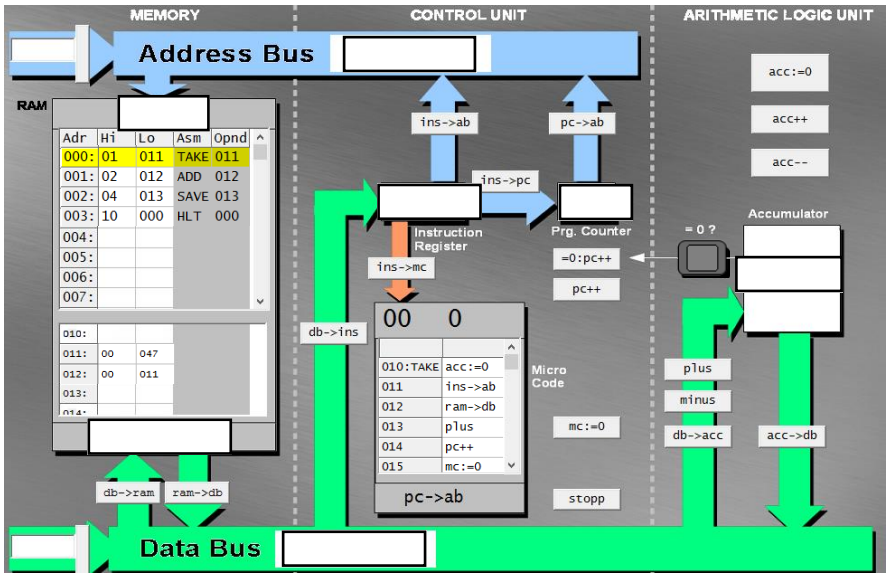
   - Folgende Assemblerbefehle wirst du noch benötigen:
     - `ADD` kennst du schon.
     - `SAVE`: Der Inhalt des Akkumulators wird in eine absolut adressierte Speicherzelle geladen.
     - `HLT`: Es wird eine Meldung angezeigt, dass das Programm abgearbeitet ist. Läuft der Simulator im Dauerbetrieb, wird der Lauf unterbrochen.
9. Gib die Befehle und Daten in den Arbeitsspeicher ein.
10. Starte das Programm mit der Schaltfläche ⏩.
    Mit der Schaltfläche ▶ werden die Befehle einzeln abgearbeitet.

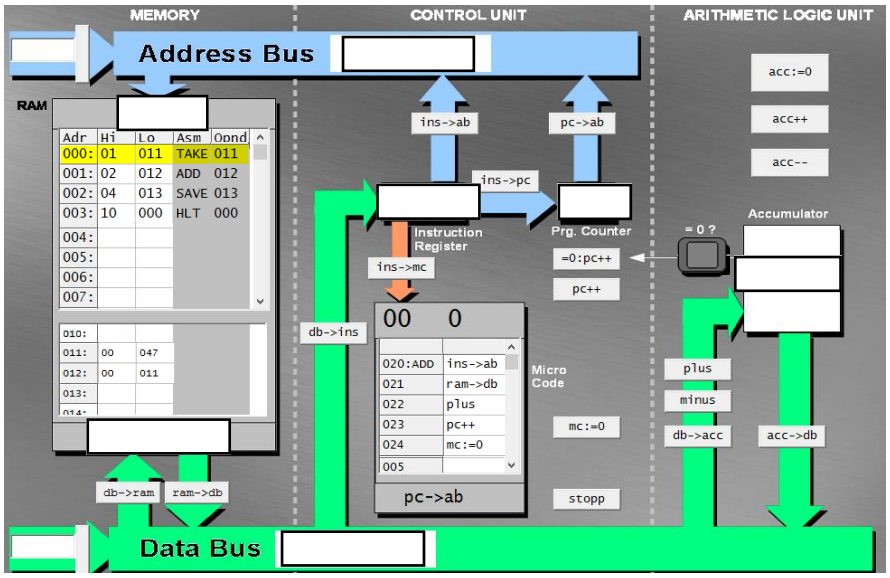
Der Speicher sollte nach dem Programmablauf die rechts abgebildeten Werte enthalten
Nachdem der Befehl `HLT` erreicht wurde,
gibt Johnny die folgende Meldung aus:

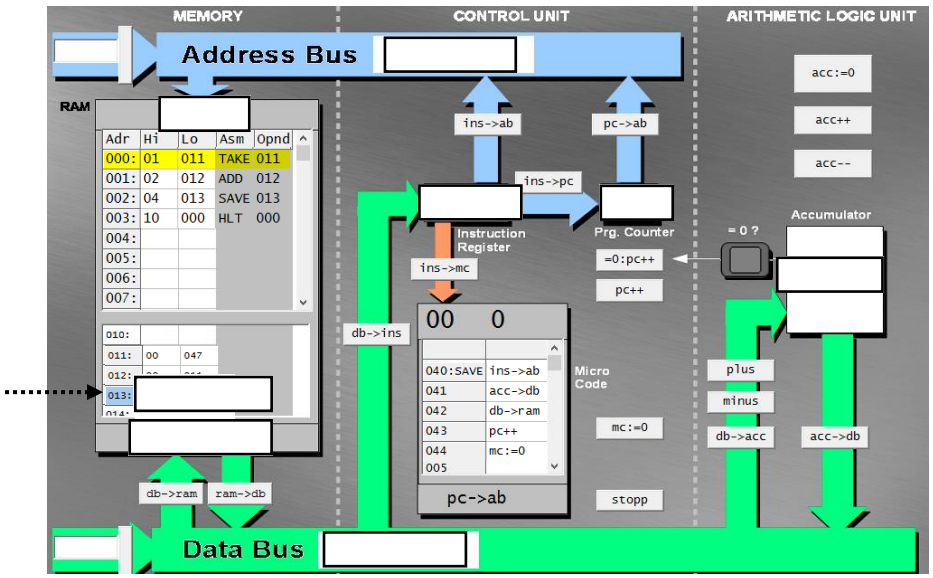


Adr	Hi	Lo	Asm	Opnd
000:	01	011	TAKE	011
001:	02	012	ADD	012
002:	04	013	SAVE	013
003:	10	000	HLT	000
004:	00	000		
005:	00	000		
006:	00	000		
007:	00	000		
010:	00	000		
011:	00	047		
012:	00	011		
013:	00	058		
014:	00	000		

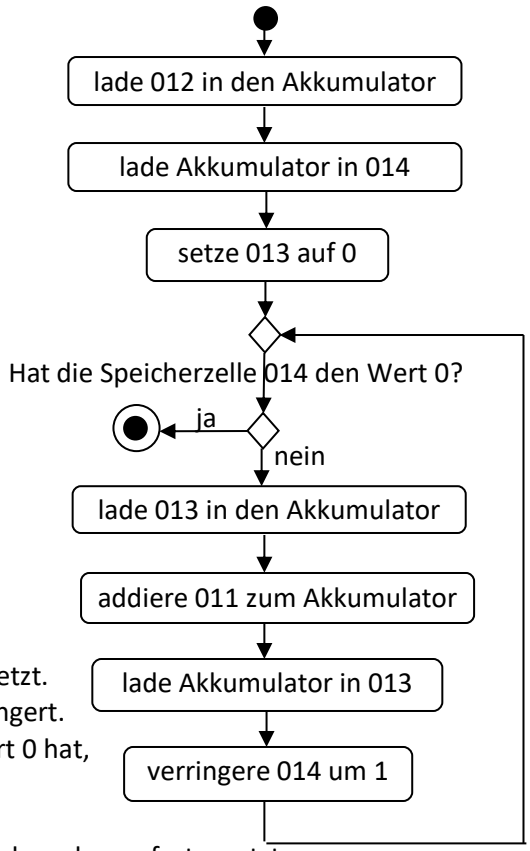
11.Trage in den folgenden Johnny-Fenstern die Daten bei der Rechnung 47 + 11 Schritt für Schritt ein.



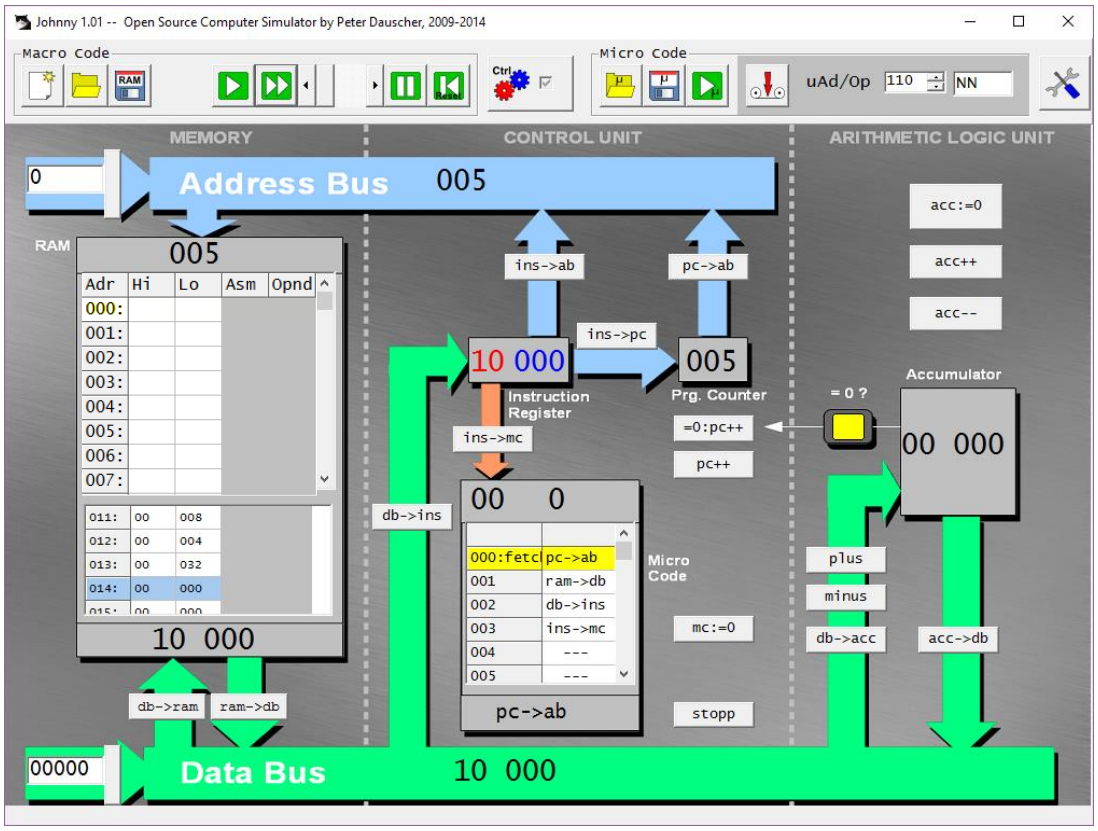




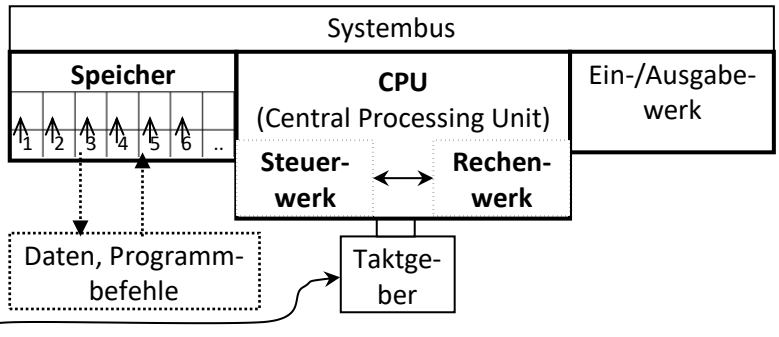
12. Beende Johnny und starten ihn neu. Erstelle dann ein Assembler- Programm, das die beiden Zahlen in den Speicherzellen 011 und 012 multipliziert und das Ergebnis in die Speicherzelle 013 schreibt.
Anmerkung zum Aktivitätsdiagramm: Die Formulierungen „den Inhalt von...“ und „...in die Speicherzelle“ werden zur kürzeren Darstellung weggelassen.
Hinweise:
   - Eine Multiplikation kann durch wiederholte Addition einer Zahl erreicht werden. Dazu wird einer der beiden Faktoren in die Speicherzelle 014 kopiert und anschließend bei jeder Addition dessen Wert um eins verringert.
   - Wenn du für Programmtests neu starten musst: Das *Befehlsregister* und den *Programmzähler* durch Klick auf `RESET` in der Werkzeugleiste löschen.
   - Beginne z. B. mit den Faktoren 8 und 4. Teste das Programm dann auch mit anderen Faktoren, insbesondere auch mit der 0 und der 1.
   - Folgende Assemblerbefehle wirst du noch benötigen:
     - `NULL`: Der Inhalt einer Speicherzelle wird auf 0 gesetzt.
     - `DEC`: Der Inhalt einer Speicherzelle wird um 1 verringert.
     - `TST`: Wenn die angegebene Speicherstelle den Wert 0 hat, dann wird bei der Ausführung des Programms eine Speicherstelle übersprungen.
     - `JMP`: Das Programm wird an der angegebenen Speicheradresse fortgesetzt.



Das Programmfenster nach dem Ablauf:



13. Erläutere die Bestandteile der Von-Neumann-Architektur. Welche Komponenten fehlen in „Johnny“, der Simulation eines von-Neumann-Rechners?



- Der **Taktgeber** fehlt – er gibt elektrische Impulse in einer bestimmten Frequenz ab. Dadurch löst er die Schaltvorgänge aus.
- Das Steuerwerk und das Rechenwerk sind in der **CPU** zusammengefasst.
  - Im **Akkumulator**
  - Das **Steuerwerk**
- **Systembus** (Gesamtheit der Verbindungsleitungen):
  - Über den **Adressbus**
- Der **Speicher** (RAM – sinngemäß übersetzt
- Bei Johnny fehlt außerdem

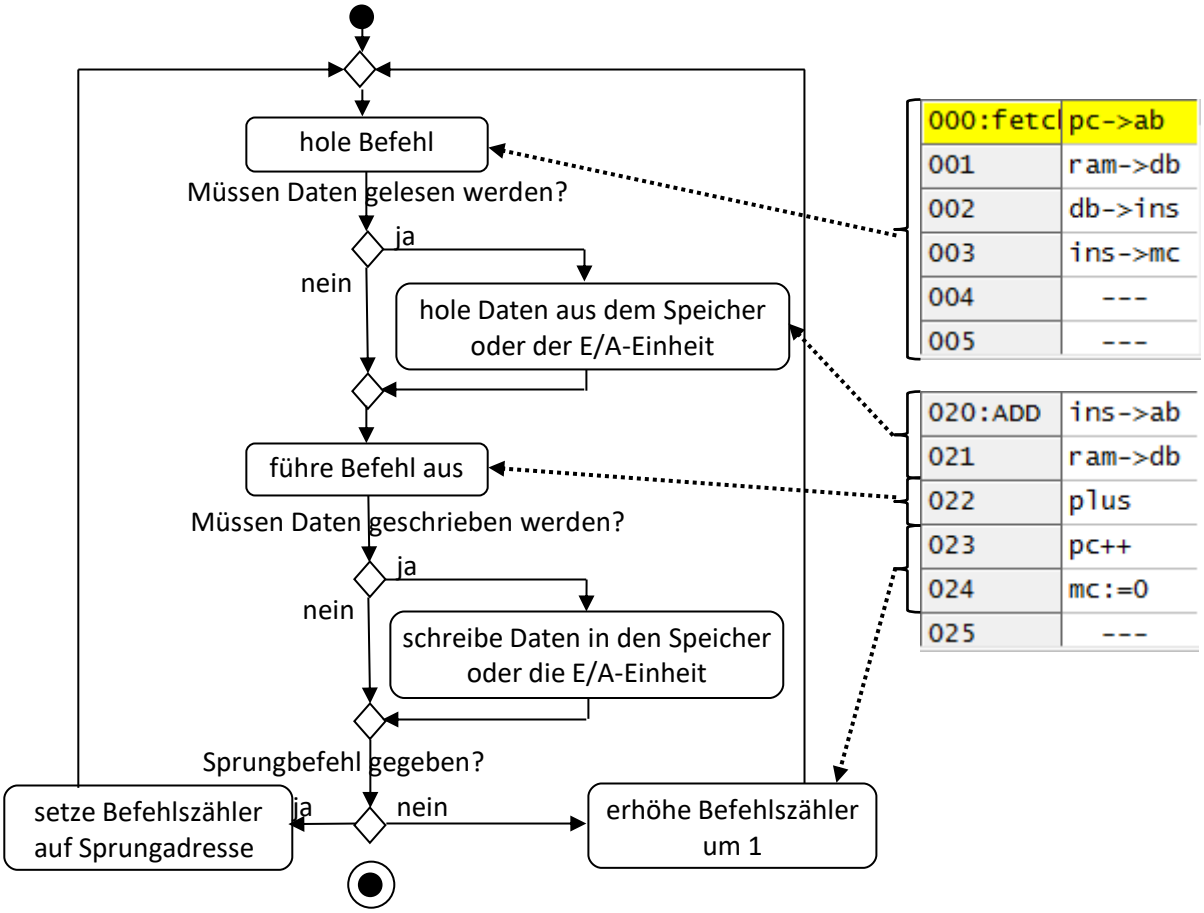
  Bei Johnny handelt es sich um ein vereinfachtes Modell eines Von-Neumann-Rechners. Ein echter Prozessor braucht aber noch Befehle wie z. B.
  - „lade Inhalt des Eingabewerks in den Datenbus“, um z. B. Tastatureingaben zu ermöglichen.
  - „lade den Inhalt des Datenbusses in das Ausgabewerk“, um Daten z. B. am Bildschirm anzuzeigen.

  Der Befehlssatz echter CPUs ist wesentlich umfangreicher. Z. B. der weit verbreitete X86-Befehlssatz umfasst über 100 Anweisungen. Er wurde 1978 mit der CPU Intel 8086 im IBM PC eingeführt und wird nach wie vor von den Intel- und AMD-Prozessoren für PCs unterstützt (Stand Oktober 2016).

14. Ergänze anhand des letzten Programmbeispiels zur Multiplikation zweier Zahlen die grundlegenden Operationen, die ein Prozessor ausführen kann.
    Bisher wurde festgehalten, dass ein Prozessor **addieren** und **vergleichen** können muss. Darüber hinaus muss er auch noch
    - einen **Wert** von dem **Speicher**
    - einen **Wert** von dem **Akkumulator**
    - zu einer beliebigen Speicheradresse

```
000: TAKE 012
001: SAVE 014
002: NULL 013
003: TST 014
004: JMP 006
005: HLT 000
006: TAKE 013
007: ADD 011
008: SAVE 013
009: DEC 014
010: JMP 003
```

15. In der Grafik sind die Mikrobefehle den Aktivitäten zu dem Befehlszyklus in einem Von-Neumann-Rechner zugeordnet.



16. Erläutere die Ausführung der folgenden Mikrobefehle.

Mikrobefehl `fetch`:

`ram->db`:

`db->ins`:

`ins->mc`:

Mikrobefehl `ADD`:

`ins->ab`:

`ram->db`:

`plus`: