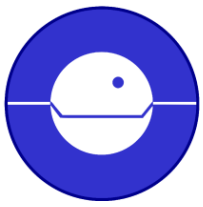
Deutsche JuniorAkademien

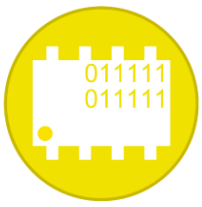
JuniorAkademie Adelsheim

13. Science Academy Baden-Württemberg 2015



Astronomie

Bodenkunde



Digitaltechnik

Kulturgeschichte/
Medienwissenschaft

Physik

TheoPrax

H.W. & J. Hector Stiftung

Regierungspräsidium Karlsruhe
Abteilung 7 – Schule und Bildung

Stiftung Bildung und Jugend

Dokumentation der JuniorAkademie Adelsheim 2015

13. Science Academy Baden-Württemberg

Träger und Veranstalter der JuniorAkademie Adelsheim 2015:
Regierungspräsidium Karlsruhe
Abteilung 7 –Schule und Bildung–
Hebelstr. 2
76133 Karlsruhe
Tel.: (0721) 926 4454
Fax.: (0721) 933 40270
E-Mail: georg.wilke@scienceacademy.de
petra.zachmann@scienceacademy.de
www.scienceacademy.de

Die in dieser Dokumentation enthaltenen Texte wurden von den Kurs- und Akademieleitern sowie den Teilnehmern der 13. JuniorAkademie Adelsheim 2015 erstellt. Anschließend wurde das Dokument mit Hilfe von LaTeX gesetzt.

Gesamtredaktion und Layout: Jörg Richter
Druck und Bindung: RTB Reprotechnkik Bensheim

Vorwort

Dieses Jahr fanden sich wieder 72 Schülerinnen und Schüler sowie Leiter, Mentoren und die Leitung zur mittlerweile 13. JuniorAkademie Baden-Württemberg in Adelsheim ein.

Die Akademie beginnt mit dem Eröffnungswochenende und findet durch das Schreiben der Dokumentation an einem Wochenende im Herbst ihren Abschluss. Im Sommer nennen wir zwei Wochen lang das Landesschulzentrum für Umwelterziehung auf dem Eckenberg unser Zuhause.

Zwischen dem Eröffnungswochenende und dem Dokumentationswochenende durchleben die Teilnehmerinnen und Teilnehmer eine Entwicklung nicht nur in fachlicher, sondern auch in persönlicher Hinsicht. Sie bekommen einen Einblick in wissenschaftliches Arbeiten und setzen sich intensiv mit ihrem Kursthema auseinander. Die Arbeit im Kurs stellt für sie eine Herausforderung dar, an der ihre Persönlichkeit reift.

Während des Sommers wachsen die Teilnehmerinnen und Teilnehmer zu einer großen Gemeinschaft zusammen. Auf dem Campus herrscht eine unbeschreibliche Atmosphäre, die einen durch die Akademiezeit trägt.

Symbolisiert werden diese Entwicklungen durch ein Motto. In diesem Jahr betrachteten wir einen Baum, der für verschiedene Aspekte der Akademie steht. Am Jahresanfang ist der Baum noch kahl. Für die Teilnehmer ist alles neu und unbekannt, und sie kennen sich noch nicht. Indem sie sich auf ihre Weise im Kurs oder bei kursübergreifenden Angeboten engagieren und die Akademie gestalten, geben sie dem Akademiebaum Nährstoffe, sodass er Blätter, Äste und Früchte bilden kann. Diese können wir ernten und mit in die Zukunft nehmen. Die geschlossenen Freundschaften, neuen Interessen und schönen Erinnerungen werden uns noch lange prägen.

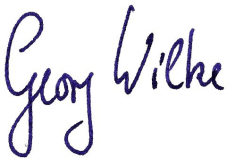
Während der Akademie begleitete uns ein Baum aus Holz, an den Erlebnisse angepinnt werden konnten. Um den Akademiebaum zum Leben zu erwecken, wurde am Dokumentationswochenende ein Mispelbaum auf dem Eckenberg gepflanzt. Der „reale“ Baum wird wachsen, Wurzeln schlagen und Früchte tragen – selbst wenn die gemeinsame Zeit zu Ende ist. Auch für die folgenden Akademiegenerationen wird der Baum sich weiterentwickeln – und wer weiß, vielleicht treffen wir einen von euch dort wieder.

Aber jetzt wünschen wir euch viel Spaß beim Lesen, Schmökern und Erinnern!

Eure/Ihre Akademieleitung

Anna Kandziora (Assistenz)

Maybritt Schillinger (Assistenz)



Georg Wilke

Dr. Petra Zachmann

Inhaltsverzeichnis

VORWORT **3**

KURS 1 – ASTRONOMIE **7**

KURS 2 – BODEN **27**

KURS 3 – DIGITALTECHNIK **55**

KURS 4 – MEDIEN **71**

KURS 5 – PHYSIK **97**

KURS 6 – THEOPRAX **117**

KÜAS – KURSÜBERGREIFENDE ANGEBOTE **131**

DANKSAGUNG **147**

Digitaltechnik – Wir bauen einen Computer

Vorwort

Michael Mattes, Bernhard Petzold, Johanna Rettenmeier

Die zwei Akademiewochen gingen schneller vorbei als wir schauen konnten. Kursinhalte, die an Schulen und Hochschulen auf ein ganzes Jahr verteilt werden könnten, haben wir in zwei Wochen angerissen. Das ist nur mit Teilnehmerinnen und Teilnehmern möglich, die außerordentlich motiviert und leistungsbereit sind.

Wir hatten uns viel vorgenommen. In den zwei Wochen wollten wir von den kleinsten Komponenten eines Computers und deren Zusammenspiel bis zur Programmierung eines selbst entwickelten PCs in einer Hochsprache gehen.

Wie uns das gelungen ist, kann man in dieser Dokumentation unserer Teilnehmerinnen und Teilnehmer lesen. Wir übergeben stolz das Wort an den Digitaltechnik-Kurs der Science Academy Baden-Württemberg 2015.

Unser Kurs

Armin fiel durch seine tollen Ideen beim Programmieren auf – selbst Michael war ganz baff, als er sah, was Armin erreicht hatte. Mit Claudius plünderte er in den Pausen die Süßigkeitenvorräte und gemeinsam mit Leo und Claudius bildete er ein effektives Team.

Claudius Wenn Claudius in der Pause zur Versorgungsstation geht, bleibt selten etwas übrig. Er ist stets hilfsbereit und zuverlässig. Während des Kurses ist er vor allem bei dem Bau der Hardware immer einer der Ersten. Außerdem baute er sich zusammen mit Armin einen eigenen versteckten Coca Cola Light Vorrat für Pausen auf.

Dominik hat immer seinen eigenen Kopf. Wenn Michael sagte, irgendetwas sei unmöglich, musste er es natürlich trotzdem

ausprobieren. Irgendwann fiel ihm dann auf, dass es echt nicht ging und Michael Recht hatte. Er war hinterher aber trotzdem einer der Schnellsten. Ihn konnte man immer fragen, wenn man bei etwas einmal nicht weiterkam. Er war zumindest am ersten Tag in der Akademie bekannt als „der, der nicht da ist“. Da er erst am Samstagabend ankam, konnten er und Sonia ihre Präsentation erst am Sonntag halten.

Elias lässt sich einfach für alles begeistern! Egal ob er den neu-erstellten Ordner von Michael sehen kann oder seine Kommentare auf dem EtherPad auch mit seiner Farbe markiert werden, alles wird mit einem „WOW“ oder einem „Oh Mein Gott“ kommentiert. Er brachte uns alle mit seinen Programmierkünsten vor allem beim Erstellen von Kurslogos weiter und auch in den KüA-Schienen begeisterte er uns immer wieder mit seinen lustigen und abwechslungsreichen KüAs, die allesamt sehr gut vorbereitet waren.

Florian ist immer gut drauf und arbeitet mit allen gut in einem Team zusammen. Er hilft, wenn er kann und gab dem Begriff Teamwork zusammen mit Marlene und Ranran eine ganz neue Bedeutung. Auch mit Claudius, Armin und Leo arbeitet er gut in einem Team zusammen und bringt oft wichtige und hilfreiche Ideen. Auch wenn er und sein Team lange an einem Problem hängen, gibt er nicht auf sondern versucht es mit neuen Lösungsansätzten (bei denen er auch ggf. sein Team erweitert). Durch seine aufgeweckte und lustige Art bringt er uns immer wieder zum Lachen.

Leo „Wir machen Apple Konkurrenz!“ Mit diesem Motto haben wir unser Kursziel erreicht und zusätzlich noch das Sportfest gewonnen. Ohne Leo ist unser Team kein Team. Er sorgt dafür, dass wir immer erfolgreich arbeiten und während des Kurses etwas zu lachen haben. Mit seiner lustigen und netten Art hat er sich mit allen in der Akademie angefreundet. In den zwei Wochen ist er zudem noch verschiedenen Mysterien der Akademie auf den Grund gegangen.

Marlene ist immer mit Geist und Seele beim Thema dabei und gibt selbst bei schwierigen Problemen nie auf. Auch kann man mit ihr über alles reden, sodass es immer sehr viel Spaß macht, mit ihr zusammenzuarbeiten. Außerdem ist sie für eine Futterpause und eine Umarmung immer zu haben und ist somit unser Knuddelmonster. Und nicht zu vergessen, ihr Lieblingsspruch: „Ernsthaft?!?“

Nicole arbeitet mit Ranran und Marlene immer so lange an Problemen, bis der Hardwaresimulator die richtigen Ergebnisse durchrattert. Sie ist immer gut gelaunt und aufmerksam im Kurs. In ihrer Zusammenarbeit mit Ranran hat sie viel Spaß und macht viele witzige Dinge. Nicole lacht insgesamt viel und steckt damit andere Personen an, wodurch ein sehr angenehmes Klima entsteht.

Nelson ist ein ruhiger ,aber charmanter Typ mit geheimen Talenten. Ein Fels in der Brandung für unsere Truppe. Man kann mit ihm auch anspruchsvollere Fragen diskutieren. Er gehört immer zu den Ersten, die mit einer Aufgabe fertig sind und hilft dann auch anderen. Zusammen mit Elias bildet er ein hervorragendes Team, das selbst schwierige Aufgaben in Rekordzeit löst.

Ranran hat für alle ein offenes Ohr; mit ihr kann man interessante und tiefgründige Gespräche führen. Sie kann immer den Aufgaben folgen, und vor allem kann man mit ihr gut im Team arbeiten. Sie ist immer offen für Fragen und das Knobeln mit ihr macht sehr viel Spaß, denn sie ist jeden Tag gut gelaunt und kann immer viel zur Lösung beitragen. In den KüA-Schienen begeisterte sie alle mit ihrer Zeichen-KüA.

Sonia Mit ihrer unbändigen Begeisterung für Sägemehlfabriken brachte sie uns schon beim EWE zum Lachen. Sie arbeitet immer mit voller Konzentration und kommt schnell zu richtigen Ergebnissen. Sie ist außerdem unser wandelnder Duden und begeisterte uns immer mit ihrer destruktiven Kritik.

Thanush ist ein sehr ruhiger Zeitgenosse, der

gegenüber den anderen Kursteilnehmern immer sehr freundlich ist. Das Spiel DigiMind bereitete ihm große Freude, und als wir dieses Spiel erstellten, trug er einen großen Teil zum Ganzen bei.

Michael, unser vergnügter Kursleiter, steckt voller Überraschungen. Er kann nicht nur selbst Honig herstellen (lassen) und seine Teilnehmer damit verköstigen. Auch „Laugaweggla“ backen gehört zu seinem Spezialgebiet. Diese Kunst durften wir in seiner Laugaweggla-KüA erlernen. Er hält uns zudem immer mit neuen Rätseln auf Trab. Wer versucht, frech zu sein, muss sich direkt einen dummen Spruch von ihm anhören.

Bernhards Lebensgeschichte passt nicht in fünf Minuten, wie wir im Kurs erstaunt feststellen mussten. Er half uns beim Programmieren und erklärte uns immer sofort Dinge, die wir nicht verstanden. Außerdem hat er das Schnäpperle raus, wie Maybritt eines Tages im Morgenplenum ankündigte.

Johanna ist unsere motivierte Schülermentorin. Sie sorgt mit Süßkram und vielen interessanten Spielen immer dafür, dass wir motiviert bleiben und uns erholt wieder in unsere Kursarbeit vertiefen können. Sie ist die gute Seele des Kurses, wenn etwas benötigt wird, egal ob Stifte, Süßigkeiten oder eine saubere Tafel, besorgt sie es sofort. Zudem bringt sie auch mal unsere Köpfe zum Rauchen, indem sie uns schwierige Rätsel stellt.

Einleitung

Die anderen Akademieteilnehmer reagierten ungläubig, als sie hörten, dass wir in nur zwei Wochen einen ganzen Computer von Grund auf bauen wollten. Wir simulierten gewissermaßen einen Computer – am Computer. Dies funktioniert so ähnlich wie bei einer Simulation in einem Computerspiel. Beispielsweise tut man bei einem Omnibus-Simulator so, als ob man mit einem Bus durch eine Stadt fährt, dabei sitzt man nur in seinem Zimmer und bedient ein paar Tasten. Ein Programm lässt es dann so aussehen, als ob man durch eine Stadt fährt. So ähnlich ist es auch beim Simulieren des Computers. Man baut den Computer nicht in Wirklichkeit auf, sondern man tippt „nur ein paar Tasten“ und ein Computerprogramm macht dann daraus einen Computer-am-Computer. Der Vorteil der Simulation besteht darin, dass Bestandteile, die mehrmals gebraucht werden, nur einmal gebaut werden müssen und danach beliebig oft kopiert werden können. Auch uns kam das Ziel, in zwei Wochen einen kompletten PC zu bauen, am Anfang so gut wie unerreichbar vor. Dennoch haben wir uns dieser Herausforderung gestellt und unser Ziel am Ende erreicht. Nachdem wir alle Bestandteile des PCs korrekt verknüpft hatten, begannen wir diesen auch zu programmieren. Zuerst in Assembly, später mit Hilfe eines Betriebssystems und der Hochsprache Jack. Das Ergebnis des Kurses war ein selbst entwickeltes Spiel namens DigiMind. Die Arbeitsatmosphäre war durchgehend angenehm, wobei die Pausen ein wichtiger Bestandteil unseres Kurses waren. Vor allem Johanna begeisterte uns immer wieder mit neuen intellektuellen Spielen. Außerdem hatten wir immer etwas zu lachen und das Wort „Langeweile“ war uns unbekannt. Auch Nervennahrung war nie weit entfernt. Es sei denn, Claudius war in der Nähe.

Transistoren

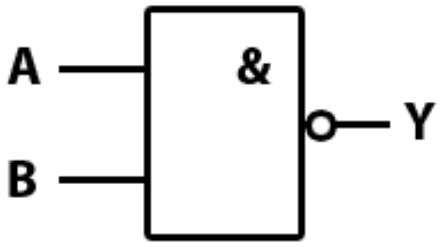
Sonia Meehan

Unserem Kurs war es sehr wichtig, dass wir von „ganz unten“, also bei den kleinsten Bestandteilen, anfangen. Um diesem Ziel gerecht zu werden, begannen wir mit Transistoren, aus denen fast alle anderen Bauteile eines Computers bestehen. Der Begriff „Transistor“ kommt aus dem Englischen und ist eine Abkürzung von „transfer resistor“. Ein Transistor besteht aus einem Siliziumkristall mit drei verschieden dotierten Schichten. Silizium ist ein Halbleiter und dotiert bedeutet, dass das Silizium leitfähiger gemacht wurde, indem das reine Silizium mit einzelnen Bor- oder Phosphoratomen beschossen wurde, die sich in das Siliziumgitter eingefügt haben. So können sich die Elektronen schon bei Zimmertemperatur viel freier bewegen. Wenn das Silizium mit Bor beschossen

wurde, nennt man dies p(=positiv)-dotiert, bei Phosphor heißt dies n(=negativ)-dotiert. Der Typ von Transistor, den wir hier betrachteten, besitzt eine pnp-Struktur, hat also jeweils zwei äußere p-dotierte Schichten und eine innere n-dotierte Schicht, wie bei einem Schnitzel (**P**aniert – **N**icht paniert – **P**aniert). Die Kontaktfläche zwischen Emitter und Basis muss viel kleiner sein als die zwischen Kollektor und Basis. Ist dies nicht der Fall, wird der Transistoreffekt ungleich schwächer. Es gibt auch Transistoren mit einer npn-Struktur. Vom Prinzip her verhalten sich die pnp- und die npn-Transistoren gleich, nur dass alles umgekehrt verläuft. Der Strom fließt genau anders herum. Der pnp-Transistor leitet, wenn keine Spannung anliegt; der npn leitet nur, wenn eine Spannung anliegt. Die Anschlüsse an die drei Schichten heißen Emitter E (n-dotiert), Basis B (p-dotiert) und Kollektor C (n-dotiert). Im Prinzip „entscheidet“ die Basis, ob zwischen dem Emitter und Kollektor Strom fließen kann. Wenn eine kleine Spannung zwischen Basis und Emitter angelegt wird, kann zwischen C und E bis zu 1000-mal so viel Strom fließen, wie zwischen B und E. Man kann also sagen, dass der Transistor wie eine Art Verstärker funktioniert, bei dem ein kleiner Basisstrom einen großen Kollektorstrom an- oder ausmacht. Somit lässt sich ein Transistor als elektrisch betriebener Stromschalter einsetzen.

Nand

Florian Kandra

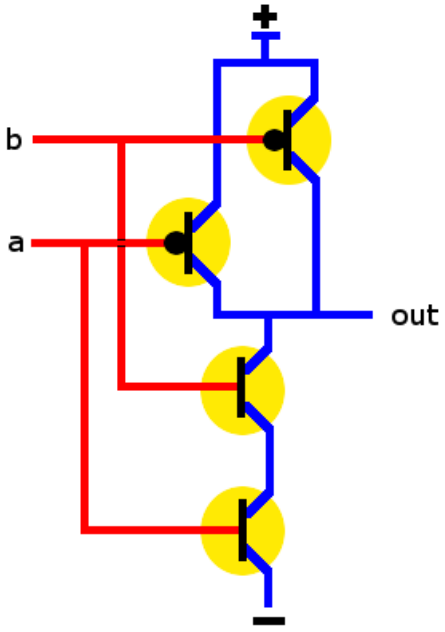


Symbol eines Nand-Gatters

Beim Nand (aus dem Englischen: „not and“, oder auf Deutsch „nicht und“) handelt es sich um den Grundbaustein des gesamten Computers. Daraus kann man alle weiteren Bausteine eines Prozessors zusammenbauen. Das Nand hat zwei Eingänge und einen Ausgang. Der Ausgang gibt genau dann eine Null aus, wenn in beide Eingänge eine Eins eingegeben wird. In allen anderen Fällen gibt der Ausgang eine Eins aus.

B	A	OUT
0	0	1
0	1	1
1	0	1
1	1	0

Wahrheitstabelle eines NAND



NAND aus Transistoren

In Wirklichkeit kann man natürlich keine Einsen und Nullen in die Eingänge eingeben, sondern man legt an die Eingänge Spannung (für eine Eins) oder keine Spannung (für eine Null) an. Ein Nand besteht aus vier Transistoren, von denen jeweils zwei mit einem Eingang verbunden sind (in der Abbildung sind die Eingänge mit a und b und der Ausgang mit out gekennzeichnet). In der Abbildung sind die pnp-Transistoren an einem Punkt zu erkennen, wohingegen die npn-Transistoren keinen Punkt davor haben. Wenn jetzt an beiden Eingängen eine Eins eingegeben oder eben Spannung angelegt wird, besteht eine Verbindung zur Masse, wodurch der Ausgang eine Null aus-

gibt. Andernfalls ist die Verbindung zur Masse unterbrochen und der Ausgang liegt auf Pluspotential bzw. der Ausgang gibt eine Eins aus.

HDL-Dateien

Nicole Anton George

HDL kann man zwar als Abkürzung für „Hab dich lieb“ sehen, aber bei der HDL, mit der wir gearbeitet haben, steht die Abkürzung für „Hardware-Description-Language“. Das heißt auf deutsch soviel wie „Hardware-Beschreibungssprache“. Genau das machen wir mit HDL: Wir beschreiben, wie der Computer die Bausteine miteinander verschalten soll.

Mithilfe von diesen Bausteinen kann man weitere Bausteine simulieren. Zunächst muss man einen Namen für den Chip (Baustein, den wir simulieren) finden. Hierbei kann man jeden beliebigen Namen wählen, allerdings macht es Sinn, den Namen des Bausteins zu benutzen. Als Beispiel wählen wir den And-Baustein, also heißt unser Chip „And“.

Um zu signalisieren, dass die Beschreibung anfängt, fügt man eine geschweifte Klammer ein. Danach werden die Eingänge und die Ausgänge des Chips definiert (die dann in HDL IN und OUT heißen). In unserem Beispiel sind die Eingänge a und b, der Ausgang heißt out. Als nächstes werden die PARTS, also die Bestandteile vom Chip beschrieben. Das And ist das gleiche wie ein verneintes Nand, das heißt es kommen die Parts Nand und Not vor.

Nun muss beschrieben werden, welche Eingänge und Ausgänge innerhalb der Bausteine verknüpft sind. Bei den Bausteinen allgemein heißt der erste Eingang a, der zweite Eingang b und der Ausgang vom Baustein out. Allerdings gibt es dabei Ausnahmen, so wie beim Not, bei dem man nur einen Eingang und einen Ausgang hat, weshalb der Eingang nur als „in“ bezeichnet wird und der Ausgang als „out“. Die Eingänge a und b vom Chip entsprechen den Eingängen des Bausteins Nand.

Jetzt geht es um den Ausgang des Nands. Dieser bekommt noch einen eigenen Namen, weil er nicht der Ausgang vom Chip ist, da das Ganze noch verneint werden muss. Deshalb nennen wir ihn „c“. Dieser Ausgang ist gleichzeitig der Eingang des Not-Bausteins und der Ausgang vom Not ist auch der Ausgang vom Chip And. Wie wird dies aber in HDL übertragen? Der Baustein heißt Nand, also schreibt man Nand und eine Klammer auf, in der dann die Eingänge und der Ausgang vom And-Baustein stehen. Der erster Eingang des Ands „a“ entspricht dem Eingang „a“ des Nand-Chips. Der zweiter Eingang des Ands „b“ entspricht dem Eingang „b“ des Nand-Chips. Es steht bis jetzt:

```
Nand(a=a;b=b(d.h.zweiter Eingang="b")).
```

Jetzt fehlt noch der Ausgang vom Nand, den wir oben „c“ genannt haben, also ist unser out=c. Das heißt, für unser Nand gilt:

```
Nand(a=a,b=b,out=c)
```

Damit ist das Nand abgeschlossen, allerdings muss das Ganze verneint werden. Das heißt der Baustein „Not“ wird verwendet. Der Eingang vom Not (es gibt nur einen, nämlich „in“) ist unser Ausgang vom Nand, also „c“. Der Ausgang vom Not (out) ist gleichzeitig auch der Ausgang vom gesamten „And-Chip“(„out“). Daher schreibt man in HDL:

```
Not(in=c,out=out)
```

Zum Schluss muss noch eine geschweifte Klammer zu gesetzt werden. Damit wurde der Chip And in HDL übertragen. Jetzt kann der Chip simuliert werden, und dieser Baustein kann weiterverwendet werden.

```
CHIP And {
    IN a, b;
    OUT out;

    PARTS:
       Nand(a=a,b=b,out=c);
       Not(in=c,out=out);
}
```

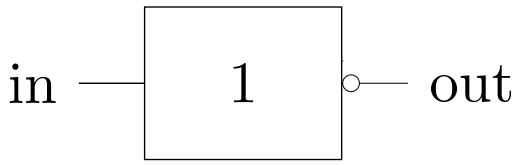
HDL-Datei des And-Chips

And, Or und Not

Claudius Berger

Aus dem schon erwähnten Baustein Nand lassen sich weitere Bausteine bauen und aus diesen immer komplexere, bis man beispielsweise einen Prozessor hat.

NOT (NICHT)



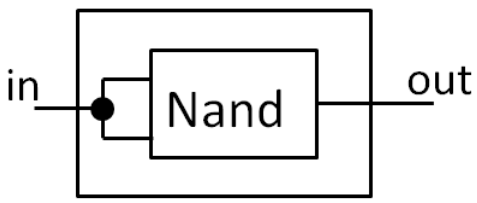
Not-Chip

Der Not-Baustein hat einen Eingang und einen Ausgang. Dieser Baustein invertiert das Signal, das heißt, wenn am Eingang eine 0 anliegt, liegt am Ausgang eine 1 an und wenn am Eingang eine 1 anliegt, liegt am Ausgang eine 0 an. Dies kann man auch an der Wahrheitstabelle erkennen.

IN	OUT
0	1
1	0

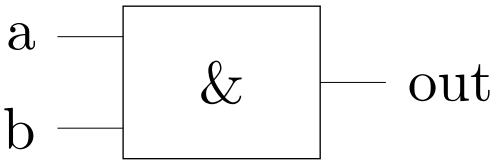
Wahrheitstabelle des Not-Chips. Links steht der Eingang, rechts der Ausgang.

Das Not besteht aus einem Nand, aber da das Not nur einen Eingang und das Nand zwei Eingänge (a und b) hat, wird das Signal, welches am Not anliegt, aufgeteilt und jeweils an die Eingänge (a und b) vom Nand angelegt. Wenn also eine 0 am Not anliegt, liegt beim Nand an beiden Eingängen eine 0 an und bei einer 1 liegt beim Nand an beiden Eingängen eine 1 an. Der Ausgang des Nands ist auch der Ausgang des Nots.



Aufbau – Not

AND (UND)



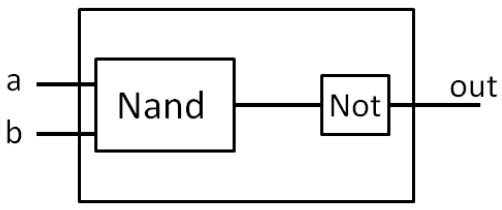
And-Chip

Der nächste Baustein ist der And-Baustein. Dieser hat zwei Eingänge (Eingang a und Eingang b) und einen Ausgang. Man könnte sich diesen Baustein wie eine Lampe vorstellen, wenn diese leuchten soll braucht man Strom UND eine Glühbirne, wenn kein oder nur eines der beiden Bestandteile vohanden ist, gibt es kein Licht, aber wenn beide Bestandteile vorhanden sind, leuchtet die Lampe. Das heißt, wenn höchstens an einem der zwei Eingängen eine 1 anliegt, liegt am Ausgang eine 0 an, wenn aber an Eingang a und an Eingang b eine 1 anliegt, liegt am Ausgang auch eine 1 an. Dies kann man auch an der Wahrheitstabelle erkennen.

B	A	OUT
0	0	0
0	1	0
1	0	0
1	1	1

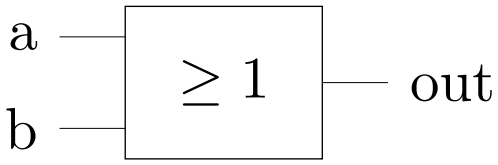
Wahrheitstabelle des And-Chips.

Das And besteht aus einem Nand und einem Not. Die Eingänge(a und b) des Ands sind mit den Eingängen(a und b) des Nands verbunden und der Ausgang des Nands führt in den Eingang eines Nots. Der Ausgang dieses Not-Bausteins ist gleichzeitig auch der Ausgang des And-Bausteins.



Aufbau – And

OR (ODER)



Or-Chip

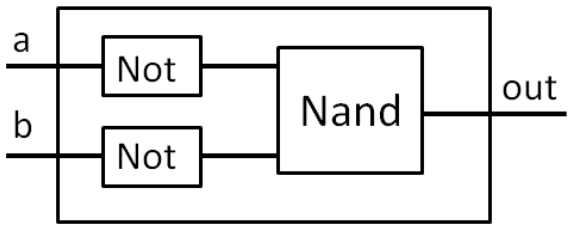
Ein weiterer Baustein ist der Or-Baustein. Er hat auch zwei Eingänge (a und b) und einen Ausgang. Diesen Baustein könnte man mit einem Bezahlvorgang vergleichen: Da man mit Bargeld ODER mit einer Kreditkarte bezahlen kann, reicht es, wenn mindestens eines dieser Dinge vorhanden ist. Das bedeutet, wenn an mindestens einem Eingang eine 1 anliegt, liegt auch eine 1 am Ausgang an. Andernfalls liegt eine 0 am Ausgang an.

Dies kann man auch an der Wahrheitstabelle erkennen.

B	A	OUT
0	0	0
0	1	1
1	0	1
1	1	1

Wahrheitstabelle des Or-Chips.

Der Or-Baustein besteht aus zwei Not-Bausteinen und einem Nand-Baustein. Die Eingänge des Or-Bausteins werden jeweils invertiert, also in ein Not geleitet. Die invertierten Signale führen beide aus den Not-Bausteinen in ein Nand, und der Ausgang des Nand ist auch der Ausgang des gesamten Or-Bausteins.



Aufbau – Or

Binärsystem und Addierwerk

Elias Lampert

Aus den Bausteinen And, Or und Not kann man weitere komplexere Bausteine erstellen, wie zum Beispiel ein Addierwerk. Der Computer versteht allerdings nur das Binärsystem. Somit mussten wir uns zuerst mit diesem beschäftigen. Im Binärsystem gibt es nur die Ziffern 0 und 1. Das klingt erstmal etwas verwirrend. Damit wir das Binärsystem verstehen, schauen wir uns zuerst das Dezimalsystem an. Im Dezimalsystem gibt es Einer-, Zehner-, Hunderterstellen und so weiter. Hier verzehnfacht sich der Wert pro Stelle. Im Binärsystem hingegen gibt es Einer-, Zweier-, Vierer-, Achterstellen und so weiter, also verdoppelt sich der Wert pro Stelle. Zum Beispiel entspricht die binäre Zahl 101 der dezimalen Zahl 5, da 101 eine Viererstelle (1*4), keine Zweierstelle (0*2) und eine Einerstelle hat (1*4). Die Zwischenergebnisse muss man nun addieren, um die dezimale Zahl zu erhalten:

$$1*4 + 0*2 + 1*1 = 4 + 0 + 1 = 5$$

Wir haben im Kurs einen 16-Bit-Computer simuliert. Das bedeutet, dass der Computer mit Zahlen rechnen kann, die im Binärsystem 16 Stellen benötigen. Man könnte nun meinen, dass die größte Zahl 65.535 wäre, diese entspricht der binären Zahl 1111.1111.1111.1111.

Allerdings gibt es nicht nur positive, sondern auch negative Zahlen. Die Stelle ganz links steht nämlich für das Vorzeichen. Eine 0 bedeutet, dass die Zahl positiv ist, und eine 1 bedeutet negativ. Die größte Zahl ist daher 32.767, dies entspricht der binären Zahl 0111.1111.1111.1111. Die kleinste Zahl ist −32.768, dies entspricht der binären Zahl 1000.0000.0000.0000. Hier springt der Wert von 0111.1111.1111.1111 auf 1000.0000.0000.0000. Die Zahl 1000.0000.0000.0001 wäre dann −32.767, der Wert nimmt somit also wieder ganz normal zu. Um die negativen Zahlen zu bestimmen, tut man so, als wäre die Zahl positiv, berechnet sie und zieht dann 32.768 von der Zahl ab. Die binäre Zahl 1000.0000.0001.0110 ist negativ (−32.768), hat eine 1 an der Sechzehnerstelle (1*16), eine an der Viererstelle (1*4)

und eine an der Zweierstelle (1*2). Nun müssen die Zwischenergebnisse addiert werden:

```
1*16 + 1*4 + 1*2 = 16 + 4 + 2 = 22
```

Da die Zahl negativ ist, muss man nun 32.768 davon abziehen:

```
22 - 32.768 = -32.746
```

Diese Binärzahl bedeutet dezimal also −32.746.

Bis jetzt haben wir uns erst mit der Umrechnung von binären Zahlen in Dezimalzahlen beschäftigt. Im folgenden Abschnitt setzen wir uns mit der Addition zweier Binärzahlen auseinander. Wenn zwei binäre Zahlen addiert werden sollen (beispielsweise 00111 + 00110 , dezimal: 7 + 6), funktioniert das wie im Dezimalsystem.

```
  0 0 1 1 1
+ 0 0 1 1 0
    0 1 1 0
-----------
  0 1 1 0 1
```

Binäre Addition von 7 und 6

Erst werden die Einer addiert: 1 + 0 = 1 mit Übertrag 0. Nun werden die Zweier addiert: 1 + 1 + Übertrag 0 = 0 mit Übertrag 1. Man würde erwarten, dass 1 + 1 = 2 wäre, aber im Binärsystem wird die 2 als 10 geschrieben. Somit ergibt sich das Ergebnis 0 mit Übertrag 1. Jetzt werden die Vierer addiert: 1 + 1 + Übertrag 1 = 1 mit Übertrag 1. Die dezimale 3 entspricht nämlich der binären 11. Deshalb ist das Ergebnis 1 mit Übertrag 1. Nun werden die Achter addiert: 0 + 0 + Übertrag 1 = 1 mit Übertrag 0. Somit ergibt sich für die Rechenaufgabe 00111 + 00110 das Ergebnis 01101, dezimal: 7 + 6 = 13.

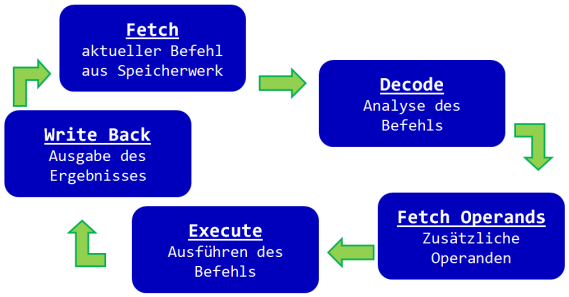
Ein Ziel des Digitaltechnik-Kurses war das Bauen eines Addierers für 16-Bit-Zahlen. Dafür braucht man zwei Arten von Addierern: den Halbaddierer und den Volladdierer. Der Halbaddierer kann zwei Zahlen addieren und dann den Übertrag bestimmen (beispielsweise 1 + 0 wäre Ergebnis 1, Übertrag 0). Der Volladdierer kann drei Zahlen addieren und dann den Übertrag bestimmen (beispielsweise 1 + 0 + 1 wäre Ergebnis 0, Übertrag 1). Der Gebrauch der Halb- und Volladdierer ist wie folgt: Man benötigt einen Halbaddierer für die Einerstelle. Hier werden nur zwei Zahlen addiert, und es gibt noch keinen Übertrag, mit dem gerechnet werden muss. Für die Zweierstelle und alle übrigen Stellen braucht man Volladdierer. Es werden die zwei jeweiligen Stellen addiert und zusätzlich und der Übertrag des letzten Ergebnisses. Somit braucht man für ein 16-Bit-Rechenwerk einen Halbaddierer und 15 Volladdierer.

Rechenwerk und Prozessor

Nelson Paraiso

Allein durch Addieren kommt man nicht weit, man muss auch andere Operationen durchführen können wie beispielsweise die Konjunktion. Hierfür gibt es die ALU. Die Arithmetic Logic Unit (ALU), auf deutsch Arithmetische und Logische Einheit, ist der Rechenkern des Computers. Sie kann zwei Zahlen addieren, substrahieren, konjungieren (UND-Verbindung), disjungieren (ODER-Verbindung) oder verneinen. Zum Ergebnis hinzu gibt es auch noch zwei weitere Outputs: Der eine gibt an, ob das Ergebnis negativ ist, der andere, ob das Ergebnis 0 ist. Die ALU hat zwei Haupteingänge für die zwei zu verrechnenden Zahlen x und y und die sechs Steuereingänge zx, nx, zy, ny, f und no, sowie einen Hauptausgang, genannt out, und zwei Nebenausgänge zr und ng. zx bzw. zy entscheidet, ob x bzw. y auf Null gesetzt wird. Wenn nx bzw. ny eins ist, wird x bzw. y verneint. f entscheidet, ob die beiden Zahlen addiert oder konjungiert werden. Wenn no eins entspricht, wird das Ergebnis verneint. Das Ergebnis wird bei out ausgegeben. Entspricht das Ergebnis Null, ist zr eins, ist es negativ, so ist ng eins.

Die Aufgabe des Prozessors, auch Central Processing Unit (CPU), zu deutsch Zentrale Ver-



Von-Neumann-Zyklus

arbeitungseinheit, ist es, zu rechnen und zu entscheiden, welche Aufgabe die nächste ist. Somit kann sie als das Gehirn des Computers angesehen werden. Der Prozessor arbeitet im Von-Neumann-Zyklus, die Schritte sind (siehe Bild):

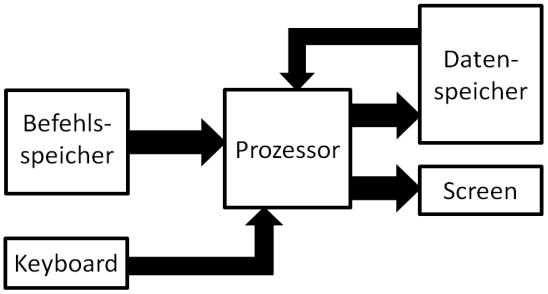
- Fetch – Holen des nächsten Befehls aus dem Befehlsspeicher
- Decode – Übersetzen des Befehls für den Rechenkern
- Fetch Operands – Holen der zusätzlichen Operanden aus dem Speicher
- Execute – Ausführen des Befehls
- Write Back – Zurückschreiben des Ergebnisses zu den jeweiligen Bauteilen

Im Prozessor sind die Hauptbestandteile deshalb eine ALU und ein PC (Program Counter – er entscheidet, welcher Befehl als nächstes ausgeführt werden soll, d. h. im Normalfall ein Befehl weiter, sonst zu dem Befehl zu dem er springen soll) enthalten, dazu kommen zwei Register, A und D, und ein Bus-System. Ein Bus-System ist die Verschaltung mehrerer Komponenten, die dieselben Leitungen benutzen. Aus der Verschaltung des Bus-Systems ergibt sich die Maschinensprache, sodass jede Computerart ihre eigene Maschinensprache hat. Es gibt einen Eingang für die Befehle, bei unserem Computer instruction genannt, und einen, der vom Memory kommt, inM genannt.

Aufbau des Computers

Marlene Esser

Auf welche Weise verarbeitet ein Computer diese ganzen Informationen? Ein einzelner Prozessor wäre wie ein Gehirn ohne Körper. Ohne Sinneswahrnehmungen und die Möglichkeit, auf diese zu reagieren, wäre unser Körper nutzlos. Das heißt auch unser Prozessor muss mit Eingängen (Inputs) und Ausgängen (Outputs) verknüpft werden.



Schematische Darstellung des Aufbaus

Ein Beispiel für einen Input ist die Tastatur (Keyboard). Bei unserer Tastatur gibt es einen Ausgang, der in die CPU führt. Die Verarbeitung der Daten aus der Tastatur beschreibt der bereits erläuterte Von-Neumann-Zyklus. Ein Output ist der Bildschirm (Screen), der die Daten der CPU auf den Bildschirm ausgibt. Diese Daten werden im Datenspeicher abgelegt. Dieser Speicher besitzt sowohl Input als auch Output, sodass man in ihm folglich Daten speichern und abrufen kann. Daten sind Texte, Zahlen, Bilder etc., die letztendlich in einer Reihe von Einsen und Nullen im Datenspeicher abgelegt werden können.

Die CPU kann entscheiden, welcher der nächste auszuführende Befehl sein soll. Diesen holt die CPU aus dem Befehlsspeicher, den der Programmierer vor dem Start des Computers gefüllt hat. Befehle sagen der CPU, welche Daten diese als nächstes bearbeiten soll. Die Befehle liegen im Maschinencode vor, der ebenfalls eine Abfolge von Einsen und Nullen ist.

Assembly/Assembler

Armin Beck

Nun kennt man den Aufbau des Computers. Um ihn allerdings zu verwenden, muss man noch mit dem Computer „kommunizieren“. Diese Kommunikation kann mithilfe von Programmiersprachen (beispielsweise Assembly) stattfinden. Assembly (auf Deutsch Assembler) ist eine Programmiersprache, die dem Maschinencode sehr ähnlich ist. In Assembly gibt es ver-

schiedene Buchstaben, mit denen man arbeiten kann. Diese stehen jeweils für einen Speicherplatz und heißen A, D und M.

Der Buchstabe A steht für das Zwischenspeicherregister mit dem Namen A-Register. Das A-Register ist der einzige Speicherplatz in diesem Programm, an dem eine Zahl eingefügt werden kann. Dies funktioniert, indem man „@x“ schreibt und nicht, wie erwartet, A=x (x steht für eine beliebige Zahl). Wenn eine neue Zahl in A gespeichert wird, geht die davor gespeicherte Zahl verloren. Beispiel: A soll Zehn sein:

```
@10   (speichere 10 in A)
```

Der Buchstabe D steht für ein Zwischenspeicherregister mit dem Namen „D-Register“. Es können auch Zahlen gespeichert werden, diese müssen jedoch erst in A gespeichert werden. Wenn man etwas Neues in das D-Register speichert, geht der vorherige Wert verloren. Beispiel: D soll 25 sein

```
@25   (speichere 25 in A)
D=A   (kopiere A (25) nach D)
```

M steht für einen Speicherchip mit dem Namen „Memory“. Dieser Speicherchip hat mehrere Stellen, an denen er etwas abspeichern kann. Wenn man nun eine Aktion mit M ausführt, wird immer die Information verwendet, die im Memory an der Stelle von A steht. A ist somit auch die Adresse von M. Wenn man mit M in Assembly arbeitet, und die Information, die in dem Memory an der Stelle Fünf steht, verwenden will, muss man erst A auf 5 setzten. Das sieht dann so aus:

```
@5    (speichere 5 in A)
M=1   (speichere 1 in Adresse 5)
```

So kann man mit der Information aus dem Memory an der Stelle Fünf arbeiten und beispielsweise eine 1 hineinschreiben.

Eine weitere wichtige Funktion im Assembly ist das Gleichheitszeichen (=). Dieses hat hier allerdings eine andere Bedeutung als in der Mathematik. In der Mathematik verwendet man ein Gleichheitszeichen, wenn zwei Ausdrücke den gleichen Wert haben. In Assembly ist dies nicht der Fall. Es wird dazu benutzt, Werte von einem in den anderen Speicherplatz zu kopieren. Hierbei wird immer der Wert aus dem Speicher rechts vom Gleichheitszeichen in den Speicher links vom Gleichheitszeichen kopiert. Der Wert des Speichers rechts vom Gleichheitszeichen bleibt gleich und der Wert des Speichers links vom Gleichheitszeichen wird überschrieben, sodass der vorherige Wert verloren geht. Beispiel:

```
D=A
```

Hierbei wird der Wert von A in D kopiert, sodass in A und D dasselbe steht und der vorherige Wert von D überschrieben wird.

Man kann allerdings nicht nur Werte von dem einen Speicherplatz in den anderen kopieren, sondern auch mit den Werten rechnen.
Beispiel:

```
D=D+A
```

Hier wird der Wert, den D zu Beginn hatte, zu dem Wert von A addiert. Mit der dabei entstehenden Summe überschreibt man D. A bleibt unverändert.

In Assembly gibt es noch weitere Befehle, mit denen man arbeiten kann. Diese sind zum Beispiel: 1 oder 0. So muss man nicht extra 1 oder 0 mithilfe des A-Registers einführen.

Somit kann man, wenn man 7 + 1 rechnen möchte, anstatt:

```
@7    (speichere 7 in A)
D=A   (kopiere A(7) nach D)
@1    (speichere 1 in A)
D=D+A (addiere D(7) und A(1)
         und speichere in D)
```

auch einfach so rechnen:

```
@7    (speichere 7 in A)
D=A+1 (addiere A(7) mit 1
      und speichere in D)
```

Programmiersprache Jack

Dominik Höfer

Nachdem wir eine Weile in Assembly programmiert hatten, merkten wir, dass diese Art des Programmierens sehr zeitaufwändig und fehleranfällig ist. Also stiegen wir auf eine Hochsprache namens Jack um. Der Vorteil dieser Hochsprache ist, dass ihre Befehle dem Englischen sehr nahe kommen, daher ist das Programmieren deutlich schneller und einfacher. In der Programmiersprache Jack gibt es viele verschiedene Befehle.

Besonders wichtig ist der if-Befehl, dieser stellt eine Art Abzweigung dar. Der Befehl wird folgendermaßen geschrieben: Er wird mit dem Wort „if“ eingeleitet, dann wird in einer runden Klammer eine Bedingung angegeben beispielsweise a>0.

Wenn diese Bedingung richtig ist, wird der erste Block von Befehlen ausgeführt, welcher in geschweiften Klammern hinter die Bedingung geschrieben wird. Wenn die Bedingung jedoch falsch ist, wird der zweite Block ausgeführt. Dieser wird ebenfalls in geschweifte Klammern geschrieben und durch das Wort „else“ (sonst, ansonsten) getrennt hinter den ersten Befehlsblock geschrieben.

Das sieht dann so aus:

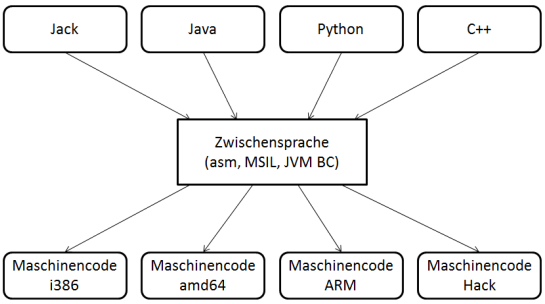
```
if (a>0) {
...
} else {
...
}
```

Ebenso gibt es einen Befehl für Schleifen, man kann Variablen definieren und ändern oder man kann Texte oder Motive auf dem Bildschirm anzeigen lassen.

Allgemein werden die Befehlsdateien von einem Compiler, zu deutsch Übersetzer für die entsprechende Hardware des Computers übersetzt. Das bedeutet, dass das übersetzte Programm nur von einem PC interpretiert werden kann. Möchten wir dasselbe Programm auf einem anderen Computer verwenden, müssen wir es zuerst speziell für diesen übersetzen.

Um zu vermeiden, dass man für jede Hardware jeweils einen Compiler zu jeder Programmiersprache schreiben muss, gibt es zwischen der Übersetzung von der Hochsprache zum Maschinencode noch weitere Zwischenschritte.

Ein Beispiel einer solchen Zwischensprache ist Assembly. Die grobe Idee dabei ist, dass man für jede Hardware nur noch einen Compiler schreiben muss, der Assembly in Maschinencode übersetzt und bei der Entwicklung einer Programmiersprache muss man nur noch einen Compiler schreiben, um die Befehle in Assembly zu übersetzen.



Grobe Funktionsweise der Übersetzung in Maschinencode

Es gibt aber außer Assembly noch viele weitere Zwischensprachen und die Übersetzung von der Hochsprache bis zum Maschinencode geht über einige weitere Schritte als nur über diesen einen.

Das Betriebssystem unseres Computers

Leopold Kloyer

Grundsätzlich versteht man unter einem Betriebssystem oder Operating System (OS) ein Kontrollzentrum, in dem es viele verschiedene Programme (Software) gibt, die einfach nur darauf achten, dass die einzelnen Hardwarekomponenten auch richtig funktionieren, wenn dies verlangt wird. Vereinfach gesagt, sind im Betriebssystem die Regeln festgehalten, die eine optimierte Zusammenarbeit der Hardwarewelt ermöglichen.

Das Betriebssystem, das wir benutzt haben, macht es uns leichter, mit der Sprache Jack

zu programmieren. Im Betriebssystem sind nämlich wichtige Befehle in Bibliotheken gespeichert. Besonders nützlich für uns waren Befehle zum Zeichnen von Linien, Rechtecken und Kreisen. Anstatt jeden Pixel auf dem Bildschirm einzeln einzufärben, können viele Pixel auf einmal definiert werden. Soll zum Beispiel ein ganzes Rechteck auf dem Bildschirm erscheinen, müssen nicht alle Punkte des Rechtecks genannt werden, sondern nur die zwei gegenüberliegende Eckpunkte. Beim Programmieren in Jack muss man dafür schreiben:

```
drawRectangle(x1,y1,x2,y2);
```

Ein Eckpunkt des Rechtecks ist (x1/y1), der andere (x2/y2).

Zusätzlich übernimmt das Betriebssystem auch die Speicherverwaltung. Als wir in Assembly einen Wert speichern wollten, benötigten wir dafür mehrere Zeilen. Das Betriebssystem vereinfacht dies, sodass dafür nur ein Befehl benötigt wird, um die Aktion auszuführen. In Jack sieht das Speichern des Wertes 10 unter der Variable x so aus:

```
var int x;
let x = 10;
```

Die Handhabung von Tastatureingaben wird durch die Betriebsystem-Bibliotheken deutlich vereinfacht. Um eine Eingabe der Tastatur zu lesen, sind nicht viele Befehle notwendig. So kann beispielsweise durch den Befehl

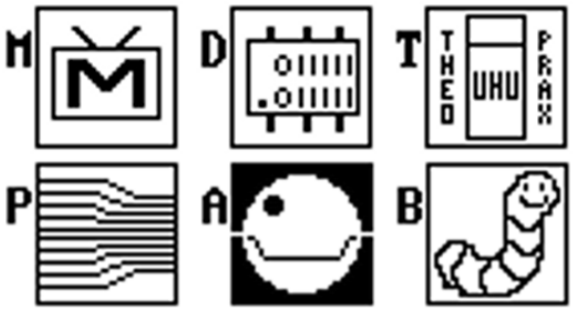
```
Keyboard.keyPressed();
```

die aktuell gedrückte Taste abgefragt werden. Zusammenfassend lässt sich sagen, dass die Funktionen der Bibliothek und damit des Betriebssystems uns sehr viel Mühe und Zeit erspart haben.

Das Spiel – DigiMind

Thanush Sivagnanalingam

Das Ziel unseres Kurses war es nicht nur einen Computer zu bauen, sondern auch ein Spiel dafür zu programmieren. Da unser Computer kein Hochleistungs-PC ist, mussten wir uns für ein einfaches Spiel entscheiden. Wir haben uns entschieden, das Spiel Mastermind auf die Akademie umzumünzen und gaben ihm den Namen „DigiMind“ (Digitaltechnik + Mastermind). Einzelne Gruppen haben dann verschiedene Bereiche des Spiels erarbeitet.



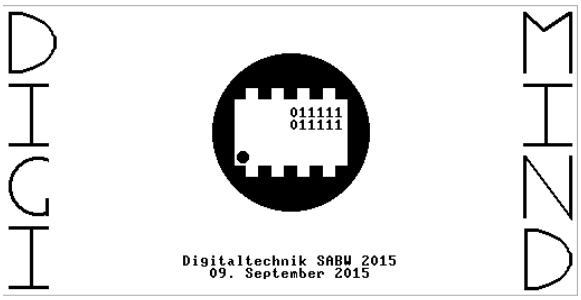
Kurslogos für DigiMind

Spielprinzip

Bei DigiMind gibt es sechs Logos (die Kurslogos der diesjährigen Akademie). Nun muss ein Spieler sich eine vierstellige geordnete Kombination ausdenken, bei der ein Logo beliebig oft und an beliebiger Stelle auftreten kann. Das Ziel eines anderen Spielers ist es dann, diese Kombination zu erraten. Wenn der Spieler ein Logo erraten hat, welches jedoch noch nicht an der richtigen Stelle ist, erscheint unter der geratenen Kombination ein Schrägstrich (/). Hat er ein Logo gefunden, das auch an der richtigen Stelle ist, erscheint ein Kreuz (X). Falls etwas nicht erraten wurde, zeigt der Computer nichts auf dem Bildschirm an. Nachdem der Spieler einen Vorschlag abgegeben hat, kann er einen nächsten Versuch starten. Er hat aber nur zehn Versuche, um die richtige Kombination zu erraten. Schafft er es nicht innerhalb der zehn Versuche, verliert er.

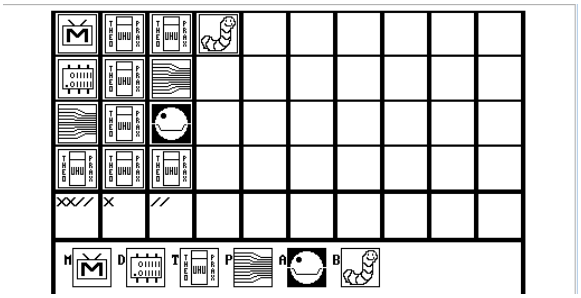
Aufbau

Das Spiel ist folgendermaßen aufgebaut: Es gibt einen Startbildschirm. Der setzt sich aus dem Kurslogo und dem dazugehörigen Text zusammen. Nachdem man auf eine Taste gedrückt hat, erscheint der Spielbildschirm. Hier gibt einer zunächst die Lösungskombination ein und danach kann der Spieler anfangen zu



Startbildschirm von DigiMind

rätseln. Gewinnt der Spieler wird er zum Endbildschirm geleitet, welcher einen fröhlichen Smiley (:-)) anzeigt. Verliert er, taucht der Endbildschirm mit dem traurigen Smiley (:-( ) auf.



Spieloberfläche von DigiMind

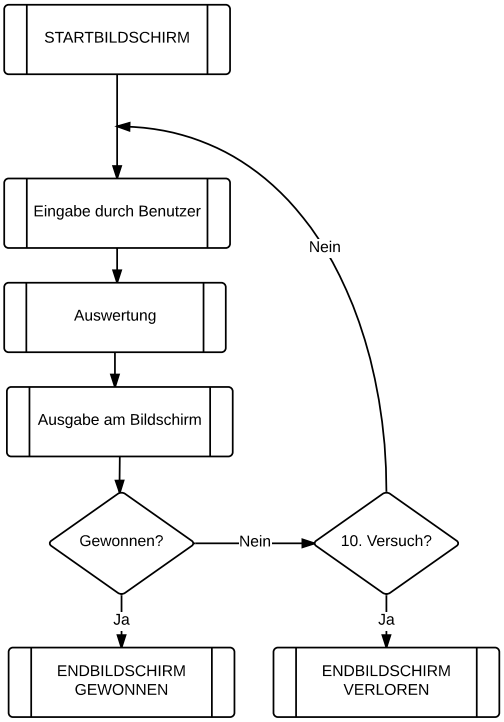
Die Logos, sowie den Start- und Endbildschirm, zu erstellen, war Aufgabe der meisten Gruppen. Da diese eigentlich Graphiken sind, kann man sich das wie ein Koordinatensystem vorstellen. Zum Beispiel hat ein Auge des Smileys die Koordinaten x=3, y=4. Dies lässt sich mit dem Befehl

```
do drawPixel(3,4);
```

einzeichnen.

Mit weiteren Funktionen wie drawLine, drawRectangle und drawCircle lassen sich komplere Bilder zusammenfügen. Auf diese Art und Weise wurden größtenteils die einzelnen Bildschirme sowie Logos entworfen. Die Logos lassen sich unter Angabe von Korrdinaten an verschiedenen Stellen auf dem Bildschirm zeichnen. Einfachheitshalber haben wir die einzelnen Graphiken zunächst in Excel „eingezeichnet“, um die einzelnen Koordinaten pixelgenau ablesen zu können. Anschließend benutzten wir die Befehle vom JackOS API, die die zuvor genannten Befehle (drawLine ...) zur Verfügung stellt, um die Logos in Programmcode umzusetzen.

Spiellogik

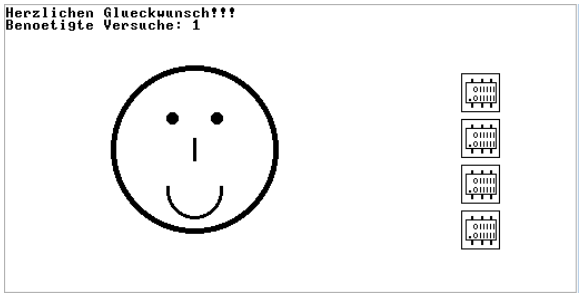


Programmablaufplan von DigiMind

Die Funktionsweise des Spiel ist in einem Programmablaufplan sowie als textuelle Beschreibung dargestellt.

Auf dem Startbildschirm wird der Name des Spiels und die sonstigen Informationen angezeigt. Durch einen Klick kommt man zum Eingabe-Bildschirm. Hier wird zunächst die Lösungskombination und dann die einzelnen Versuche des Spielers eingegeben. Danach wertet der Computer die Eingabe mit der Lösungskombination aus. Je nachdem, ob der Vorschlag mit der gesuchten Lösung übereinstimmt oder nicht, zeigt der Ausgabe-Bildschirm, dass entweder ein Logo stimmt, (Bei der richtigen Stelle mit /, bei einer falschen Stelle mit X), oder dass es falsch ist. Bei einem falschen Logo erscheint nichts auf dem Bildschirm. Währendessen überprüft der Computer noch, ob der Spieler gewonnen hat. Wenn er gewonnen hat, kommt der

Endbildschirm mit den fröhlichen Smiley ☺. Falls nicht, schaut der Computer, wie viel Versuch der Spieler noch hat. Hat er weniger als Zehn, darf er einen erneuten Vorschlag abgeben. Sind die zehn Versuch überschritten, hat der Spieler verloren und der Endbildschirm mit dem traurigen Smiley ☹ taucht auf.



Endbildschirm von DigiMind

Exkursion

Ranran Ji

Digitaltechniker auf Exkursion

Wer sich denkt, dass die Digitaltechniker ihre Zeit nur am Computer verbringen, der hat sich gewaltig getäuscht. Am Mittwoch in der ersten Akademiewoche war es endlich so weit: Die ganze Gruppe begab sich auf eine Exkursion. Und das nicht irgendwohin, sondern zu der Quelle unserer ganzen Arbeit, das Institut für Mikroelektronik Stuttgart (IMS CHIPS).

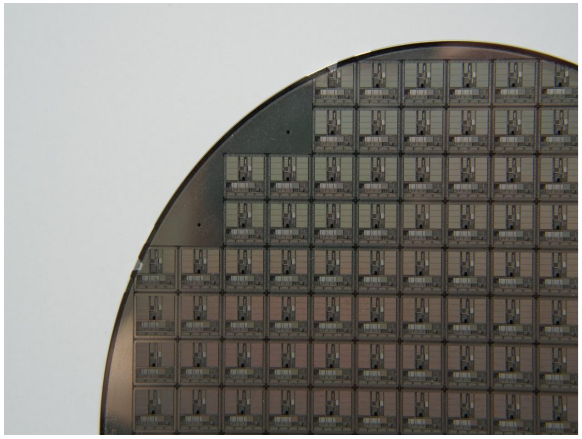
Früh am Morgen ging es nach Stuttgart und zwar mit den unterschiedlichsten Verkehrsmitteln, weil an jenem Tag der Schienenumbau die normale Zuglinie behinderte. Aber das viele Umsteigen nahmen wir in Kauf, da wir dadurch nicht nur unsere Kursteilnehmer mehr kennenlernten, sondern auch gleich die ereignisreichen Lebensgeschichten unserer beiden Kursleiter zu hören bekamen. Zusätzlich unterhielt uns Michael immer mit sehr interessanten Themen wie dem Geheimnis der Kryptographie oder verschiedene Ratespiele, über die manche sich den ganzen Tag den Kopf zerbrachen. Gegen Mittag legten wir eine Pause am Stuttgarter Hauptbahnhof ein und konnten in kleinen Gruppen entweder die Stadt erkunden oder uns etwas gegen das lästige Magenknurren holen. Die Meisten entschieden sich für die zweite Möglichkeit. Nach einer Stunde ging es dann glücklich und mit vollen Magen weiter zum eigentlichen Ziel unseres Ausflugs: Das Institut für Mikroelektronik auf dem Campus der Uni Stuttgart.

Dort erwartete man uns bereits mit einem Vortrag und einem Video über das Institut und die Herstellung von Wafern, runden Siliziumplatten, in denen die einzelnen Chips eingearbeitet werden. Das Ganze erweckte sofort bei allen die Neugier und Wissbegierde, eine gute Voraussetzung für die bevorstehende Führung. Wir wurden in zwei Gruppen eingeteilt. Die erste Gruppe ging zuerst zu den Reinräumen. Der Name sagt schon vieles darüber aus. In diesen Räumen herrschen vorgeschriebene Sauberkeitsbedingungen, um die Herstellungsprozesse nicht durch Staub oder andere Fremdpartikel, hohe Luftfeuchtigkeit oder wechselnde Temperaturen zu gefährden. Alle Angestellten in dieser Abteilung arbeiten deshalb in weißer Schutzkleidung. Nachdem sie die Haarnetze und Hausschuhe angezogen haben, müssen sie jedes Mal beim Eintreten durch eine Luftschleuse und danach noch mal durch eine zweite Schleuse gehen. Auch wir mussten uns kleine blaue Plastiktüten über die Schuhe ziehen, was alle sehr amüsant fanden. Es war beeindruckend, die Mitarbeiter in ihren Arbeitskitteln hinter den Glaßcheiben zu beobachten und dabei die einzelnen Schritte der Chipherstellung näher kennenzulernen. Da wird einem noch mehr bewußt, wie viel Arbeit hinter einem Computer steckt.

Zuerst schneidet man hauchdünne Siliziumplatten aus einem Siliziumzylinder, der eine sehr reine Struktur besitzt, heraus. Ein paar von uns durften sogar versuchen, die Spitze des Zylinders anzuheben, scheiterten jedoch kläglich,

da diese sehr schwer ist. Und das blieb nicht das einzige Mal, wo wir sprachlos staunten. Die Siliziumplatten werden anschließend abgeschliffen. Aber wie kommen nun die vielen Chips auf eine solche Platte und gar die die Millionen Transistoren auf einem Chip? – Die Lösung: Fotolackverfahren.

Dabei trägt man zunächst eine Schicht Aluminium auf die Platte auf, die sofort mit Fotolack besprüht wird. Nun legt man eine Schablone aus mit Chrom beschichtetem Glas auf den Wafer bzw. auf die Siliziumplatte und belichtet sie. Dazu arbeitet man in speziellen Räumen, die wegen der Belichtung in dunkelorangenefarbenes Licht getaucht sind. Bei dieser Belichtung wird die nicht abgedeckte Fotolackschicht beschädigt, die anschließend durch ein Säurebad abgewaschen wird. Zum Abschluß wird die Aluminiumschicht ohne den Fotolack weggeätzt. Dieses Verfahren wird mehrere Male wiederholt, bis sich daraus bis zu 15 Schichten bilden. Dadurch sind die einzelnen Verbindungen zwischen den Transistoren geschaffen. Doch die Mikrochips sind noch nicht am Ende ihrer Reise angelangt.



Chips auf einem Wafer

Mit einem Chip allein könnte man noch nicht viel anfangen. Um ihn einfach in ein Computersystem einzubauen, bräuchte man ein passendes Gehäuse. Es gibt im Allgemeinen zwei Modelle, entweder aus Plastik oder Keramik. Beide Varianten durften wir beim zweiten Teil der Führung sogar hautnah miterleben. Auch ein Wafermodell bekamen wir zu sehen. Aber zurück zum Gehäuse; die Chips werden mit feinen Bonddrähten an das Gehäuse angeschlossen. Dabei waren die Aluminiumdrähte so fein, dass wir sie kaum mit unseren Fingern erspüren konnten. Als würden sie schweben, dabei hatten sie noch einen kleineren Durchmesser als das menschliche Haar. Doch durch die Führung erhielten wir auch einen Einblick in die Arbeitsatmosphäre der Mitarbeiter. Dass der Arbeitsplatz auch ein zweites Zuhause für manche ist, sahen wir an den vielen Insider-Witzen hinter jeder Ecke, wie zum Beispiel „Karins Porsche“ an einem Transportwagen. Leider wurde am Ende die Zeit zu knapp und wir konnten nicht mehr den sicher spannenden Vortrag über Kamerachips anhören. Da der Digitaltechnikkurs im letzten Jahr den Zug verpaßt hatte und somit am Abend viel zu spät angekommen war, nahmen wir uns umso mehr vor, es dieses Jahr nicht zu wiederholen. In Joggingtempo eilten wir zurück zur S-Bahn Haltestelle. Und dieses Jahr waren wir tatsächlich pünktlich. Alle, völlig ermüdet, hofften, dass Georg uns womöglich vom Adelsheimer Bahnhof abholen könnte. Seine Antwort war nur, er werde bis zu unserer Ankunft versuchen den Berg abzuräumen, damit wir es einfacher haben. Leider schaffte er es nicht rechtzeitig. Aber das machte nichts. Digitaltechniker geben niemals auf. So schleppten wir uns auch ohne Georgs Hilfe den Berg rauf und freuten uns anschließend umso mehr auf das wohlverdiente Abendessen.

Ein großer Dank geht an das Institut für Mikroelektronik Stuttgart für diese einmalige Möglichkeit, hinter die Kulissen eines so komplexen Herrstellungsprozesses blicken zu dürfen. Ebenso an Herrn Leibold und Herrn Berndt für die tolle Führung und last but not least Herrn Strobel und Herrn Futterer.

Sportfest

Leopold Kloyer

Geprägt von größtem Teamgeist und Begeisterung gelang es dem diesjährigen Digitaltechnikkurs sich den Titel des Sportfestsiegers zu sichern und bei der Siegerehrung ganz oben zu stehen. Einen kompletten Nachmittag lang haben wir als Team für die Ehre des Digitaltechnikkurses gekämpft. Mit Leichtigkeit konnten wir die restlichen fünf Kurse in den schwierigen

Disziplinen schlagen. Das lag nicht nur daran, dass die anderen Kurse schon vor Furcht erzitterten, wenn sie unseren ohrenbetäubenden Kampschrei hörten. Auf dem gesamten Gelände des LSZU schallte es:

„Ihr seid NULL! Wir sind EINS! 111 – Wir verlieren keins!“

Und mit dieser Ansage und vor allem deren Erfüllung konnten wir wieder einmal zeigen, welche Power in Computern und deren Benutzern steckt!

Auf dem Weg zum Triumph!

Danksagung

Wir möchten uns an dieser Stelle bei denjenigen herzlich bedanken, die die 13. JuniorAkademie Adelsheim / Science-Academy Baden-Württemberg überhaupt möglich gemacht haben.

Finanziell wurde die Akademie in erster Linie durch die H. W. & J. Hector Stiftung, durch die Stiftung Bildung und Jugend sowie den Förderverein der Science-Academy unterstützt. Dafür möchten wir an dieser Stelle allen Unterstützern ganz herzlich danken.

Die Science-Academy Baden-Württemberg ist ein Projekt des Regierungspräsidiums Karlsruhe, das im Auftrag des Ministeriums für Kultus, Jugend und Sport, Baden-Württemberg und mit Unterstützung der Bildung & Begabung gGmbH Bonn für Jugendliche aus dem ganzen Bundesland realisiert wird. Wir danken daher dem Leiter der Abteilung 7 des Regierungspräsidiums Karlsruhe, Herrn Vittorio Lazaridis, der Referatsleiterin Frau Leitende Regierungsschuldirektorin Dagmar Ruder-Aichelin, Herrn Jurke und Herrn Dr. Hölz vom Ministerium für Kultus, Jugend und Sport sowie dem Koordinator der Deutschen Schüler- und JuniorAkademien in Bonn, Herrn Volker Brandt, mit seinem Team.

Wie in jedem Jahr fanden die etwas über einhundert Gäste sowohl während des Eröffnungswochenendes und des Dokumentationswochenendes als auch während der zwei Wochen im Sommer eine liebevolle Rundumversorung am Eckenberg-Gymnasium mit dem Landesschulzentrum für Umwelterziehung (LSZU) in Adelsheim. Stellvertretend für alle Mitarbeiter möchten wir uns für die Mühen, den freundlichen Empfang und den offenen Umgang mit allen bei Herrn Oberstudiendirektor Meinolf Stendebach, dem Schulleiter des Eckenberg-Gymnasiums, besonders bedanken.

Zuletzt sind aber auch die Kurs- und KüA-Leiter gemeinsam mit den Schülermentoren und der Assistenz des Leitungsteams diejenigen, die mit ihrer hingebungsvollen Arbeit das Fundament der Akademie bilden. Ein besonderer Dank gilt an dieser Stelle Jörg Richter, der auch in diesem Jahr für die Gesamterstellung der Dokumentation verantwortlich war.

Diejenigen aber, die die Akademie in jedem Jahr einzigartig werden lassen und die sie zum Leben erwecken, sind die Teilnehmerinnen und Teilnehmer. Deshalb möchten wir uns bei ihnen und ihren Eltern für ihr Vertrauen ganz herzlich bedanken.