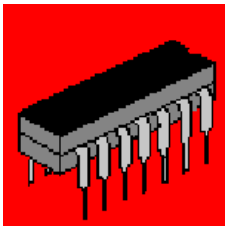
Inside CPU



Handbuch

Manfred Sommer

Die Texte basieren teilweise auf: Manfred Sommer, Bernhard Streiko, Manfred Franke: CPU-Simulation – Wie arbeitet ein Computer? Wiesbaden 1991 (Gabler Verlag) sowie auf der von Jörg Hartwich entworfenen und von Bernhard Streiko überarbeiteten Online-Hilfe. Allen an Inside CPU Mitwirkenden sei herzlich gedankt.

Stand: Mai 2001

Inhaltsverzeichnis

1 Einführung in Inside CPU

Herzlich Willkommen bei Inside CPU !

1.1 Zielsetzung

Computer sind aus dem Berufs- und Alltagsleben nicht mehr wegzudenken. Deshalb dringt die Informatik seit vielen Jahren verstärkt in Schulen und Hochschulen sowie in die berufliche Erstausbildung und Weiterbildung vor. Da immer mehr Tätigkeiten von EDV-Anwendungen durchdrungen werden, hat diese "Informatisierung" auch viele sog. Misch- und Randberufe der Datenverarbeitung erfasst. Dies hat nachhaltige Veränderungen in sehr vielen Studiengängen und Ausbildungsberufen ausgelöst, für die Informatik-Anwendungen im Vordergrund stehen.

Während vertiefte Hardwarekenntnisse für die Kernfach-Informatik an den Hochschulen und die IT-Kernberufe des Dualen Systems zum selbstverständlichen Ausbildungsinhalt zählen, stehen sie in den Studien- und Ausbildungsgängen für Informatik-Anwender nicht im Mittelpunkt des Interesses. Dennoch wird zumindest ein Grundverständnis der Arbeitsweise des Herzstücks von DV-Systemen, der sog. CPU (Central Processing Unit), nach wie vor allgemein für wichtig gehalten. Leider gestaltet sich die Vermittlung dessen, was sich innerhalb dieser "black box" abspielt, didaktisch nicht gerade einfach, weil es abgesehen von schwarzen Bauteilen auf verwirrenden Leiterplatinen wenig zu sehen gibt.

Inside CPU will diese "didaktische Lücke" schließen helfen. Es nutzt den Computer zum Lernen über Computer, ist also ein Stück Lernsoftware. Das Lernen mit dem Computer wird traditionell als CBT (Computer Based Training) oder neuerdings als E-Learning bezeichnet. Es bietet u.a. folgende Vorteile:

- Individualisierung des Lernens: Sie entscheiden, wann, wie lange und wie schnell Sie sich mit dem Stoff beschäftigen.
- Visuelle Anschaulichkeit des Lernstoffs.

Im Mittelpunkt von Inside CPU steht die Programmierung einer stark vereinfachten CPU mit einer fiktiven, maschinennahen Programmiersprache, die gleichwohl die grundsätzlichen Prinzipien und Abläufe sichtbar werden lässt. Deshalb besteht Inside CPU aus einem statischen Editor-Fenster zur Bearbeitung der Programme und aus einem dynamischen Simulationsfenster, in dem der Ablauf der selbsterstellten oder der mitgelieferten Beispielprogramme beobachtet werden kann. Inside CPU zeigt dabei den schematischen Aufbau eines Zentralprozessors und die Abarbeitung der Maschinenbefehle (Speicherbelegung und Datenfluss). Der Benutzer kann die Simulationsgeschwindigkeit in fünf Stufen variieren.

1.2 Zielgruppe

Inside CPU richtet sich an alle, die erste Eindrücke vom Innenleben eines Mikroprozessors gewinnen möchten: insbesondere an Studierende, Schüler und Auszubildende in anwen-

dungsinformatischen Studien- und Ausbildungsgängen und ihre Lehrkräfte.

1.3 Feedback und Infos im Internet

Fragen, Anregungen und Kritik richten Sie bitte an:

Prof. Dr. Manfred Sommer
Hochschule für Wirtschaft und Politik
Arbeitsgebiet Wirtschaftsinformatik
Von-Melle-Park 9
D-20146 Hamburg
E-Mail: sommer@hwp-hamburg.de

Zusätzliche Informationen und Materialien erhalten Sie im **Internet** unter:

http://www.hwp-hamburg.de/fach/fg_bwl/dozentinnen/sommer/downloads/InsideCPU

Dort steht insbesondere eine **begleitende PowerPoint-Präsentation für Lehrende** bereit (InsideCPU-Einführung.ppt bzw. InsideCPU-Einführung.pps für diejenigen, die nicht über PowerPoint verfügen).

2 Installation von Inside CPU

2.1 Installationsvoraussetzungen

Inside CPU setzt folgende Anforderungen an die Hardware und Software des PC's voraus, auf dem es installiert werden soll:

- IBM-kompatibler PC 486 (Pentium-Prozessor empfohlen)
- 32 MB RAM
- 5 MB freier Festplattenspeicher
- Bildschirmauflösung: SVGA (800x600) optimal, 600x480 oder höher als 800x600 möglich
- Windows 95 (oder höher), Windows NT 4.0 (oder höher)

2.2 Installationsprozedur

Um "Inside CPU" auf Ihrem Rechner zu installieren, benötigen Sie drei Installationsdateien: setup.exe, SETUP.LST und INSIDE_CPU.CAB. Diese werden Ihnen von ask|net unter https://www.softwarehouse.de zur Verfügung gestellt. Da Sie die Installation höchstwahrscheinlich von einer Festplatte aus vornehmen, beachten Sie bitte: das Verzeichnis, in das Sie die drei Installationsdateien kopieren, muss leer sein!

Mit dem Programm "setup.exe" wird die Installation von Inside CPU gestartet. Das Setup-Programm nimmt entsprechend Ihrer Systemkonfiguration einige Voreinstellungen vor. Diese

werden in der Datei WIN_CPU.INI gespeichert und können nach Ihren Bedürfnissen modifiziert werden. Nähere Informationen zu dieser Datei können Sie dem Anhang im Abschnitt 7.1 entnehmen.

2.3 Bekannte Probleme

Fehlermeldung "WIN_CPU.INI nicht gefunden! Programm bitte neu installieren!"

Sollte diese Fehlermeldung beim Start von Inside CPU erscheinen, ist keine Neuinstallation erforderlich. Sie müssen diese Datei lediglich mit dem Windows-Explorer suchen und in das Windows-Verzeichnis auf Ihrer Festplatte verschieben, z.B. C:\WINNT.

2.4 Deinstallation

Um Inside CPU auf Ihrem Rechner zu deinstallieren, verwenden Sie einfach "Windows-Komponenten hinzufügen/entfernen" des über die Windows-Systemsteuerung erreichbaren Dienstprogramms "Software".

3 Theoretische Grundlagen zu Inside CPU

Daten werden innerhalb des Computers und seiner Peripherie in Form von Stromimpulsen weitergeleitet. Die kleinste und einfachste Information besteht darin, dass entweder Strom fließt oder nicht fließt. Dies ist eine zweiwertige Information. Auch eine Glühbirne kann eine Informationseinheit darstellen. Bei einem geschlossenen Stromkreis fließt Strom und die Birne brennt; ist der Stromkreis geöffnet, bleibt sie dunkel. Ist die Glühbirne ausgeschaltet, kann dieser Zustand mit einer 0 kodiert werden; ist die Lampe an, schreiben wir eine 1. Auf diese Weise lassen sich mit der Glühbirne zwei verschiedene Zustände (Glühbirne brennt, Glühbirne brennt nicht) darstellen.

Andere, in der Datenverarbeitung bedeutsame technische Repräsentationen zweiwertiger Informationen sind:

Lochkarte:	Loch gestanzt / nicht gestanzt
CD-ROM:	Vertiefung gebrannt / nicht gebrannt
Magnetplatten und –bänder:	Nordpol / Südpol
Halbleiterbaustein:	leitend / nicht leitend

Zeichensysteme, die zwei Zustände annehmen können, werden als **binär** und ihre Zeichen als **Bit** (Abkürzung für binary digit) bezeichnet. Ein Binärsystem ist z.B. auch eine Fußgängerampel mit rot und grün. Als Zeichenvorrat eines Binärsystems werden üblicherweise verwendet:
Die binäre Null (0 oder L [Low])
Die binäre Eins (1 oder H [High])

Besteht das Binärsystem aus den beiden Zahlen 0 und 1, dann spricht man von einem dualen Zahlensystem oder **Dualsystem**.

Da die Informationen der realen Welt in aller Regel aus mehr als zwei verschiedenen Zeichen bestehen, müssen mehr als zwei Bits verwendet werden, um diese Zeichen binär darzustellen. Je mehr Bits für die Darstellung eines Zeichens verwendet werden, desto mehr unterschiedliche Zeichen können dargestellt werden. Reiht man 8 Bit aneinander, dann gibt es 2*2*2*2*2*2*2*2 = 2^8 = 256 Möglichkeiten für verschiedene 0/1-Kombinationen, also 256 unterscheidbare Zeichen.

In der EDV gibt es mehrere bedeutsame Konventionen für die Zusammenfassung von Bits zu größeren Informationseinheiten:
8 Bit bilden ein **Byte**.
4 Bit bilden demnach ein **Halbbyte**.
16 Bit oder 2 Byte bilden ein **Wort**.
32 Bit oder 4 Byte bilden ein **Doppelwort**.

Größere Datenvolumina lassen sich mit den folgenden **Maßeinheiten** schneller erfassen, die z.B. auch zur Kennzeichnung von Hauptspeichergrößen verwendet werden.
1.024 Byte = 1 Kilobyte (KByte)
1.024 KByte = 1 Megabyte (MByte)
1.024 MByte = 1 Gigabyte (GByte)
1.024 GByte = 1 Terabyte (TByte)

3.1 Zahlensysteme

Für die digitale Darstellung von Daten in Computern sind vor allem vier Zahlensysteme von Bedeutung, die sich durch die Wertigkeit ihrer Stellen unterscheiden:
Dualsystem (Basis = 2): Wertigkeit steigt um das 2-fache.
Oktalsystem (Basis = 8): Wertigkeit steigt um das 8-fache.
Dezimalsystem (Basis = 10): Wertigkeit steigt um das 10-fache.
Hexadezimalsystem (Basis = 16): Wertigkeit steigt um das 16-fache.

Diese Zahlensystem sind sämtlich sogenannte Stellenwertsysteme, die sich dadurch auszeichnen, dass ihr Zahlenwert z gleich Summe der Produkte aus dem Nennwert (a_k) der einzelnen Ziffern und dem Stellenwert (B^k) innerhalb der Zahl ist:

$$z = \sum a_k * B^k$$

Basis B der Zahlen	**Nennwerte a der Ziffern**
Dualsystem (Basis = 2)	0, 1
Oktalsystem (Basis = 8):	0, 1, 2, 3, 4, 5, 6, 7
Dezimalsystem (Basis = 10):	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Hexadezimalsystem (Basis = 16):	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 (A), 11 (B), 12 (C), 13 (D), 14 (E), 15 (F)

Zahlensysteme müssen nicht zwingend Stellwertsysteme sein. Das Römische Zahlensystem war z.B. kein Stellenwertsystem. Näheres entnehmen Sie bitte den PowerPoint-Folien.

Dezimalzahlen

Dieses Zahlensystem benutzt die Ziffern 0 bis 9. Der Stellenwert einer Dezimalzahl wächst um das 10-fache. Es ist das im Alltag weltweit gebräuchlichste Stellenwertsystem.

Die Darstellung von Dezimalzahlen größer als 255 bzw. kleiner als –128 im Computer wird durch die Verwendung mehrerer Bytes für eine Zahl realisiert. Mit zwei Bytes können kleine natürliche Dezimalzahlen von 0 bis 65535 bzw. ganze Zahlen von –32768 bis 32767 dargestellt werden.

Dualzahlen

Das Dualsystem verwendet lediglich die Ziffern 0 und 1. Der Stellenwert einer (echten) Dualzahl wächst um das 2-fache. Den Dezimalzahlen 0 bis 15 entsprechen folgende Dualzahlen.

Dualzahl				Dezimalzahl
2^3	2^2	2^1	2^0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

Beispiel: $10010111_2 = 128 + 16 + 4 + 2 + 1 = 151_{10}$

2^k	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	128	64	32	16	8	4	2	1

a_k	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**
$a_k * 2^k$	128	0	0	16	0	4	2	1

Oktalzahlen

Genauso wie es möglich ist, ein Zahlsystem auf den Ziffern 0 und 1 aufzubauen, kann man auch auf anderen Ziffernfolgen Zahlensysteme aufbauen. Die Zahlen aus dem Zahlensystem zur Basis 8 heißen Oktalzahlen. Die Oktalzahlen benutzen nur die Ziffern von 0 bis 7. Genau wie im Dezimalsystem werden beim Zählen die Ziffern der niedrigsten Stelle solange erhöht, bis

die höchste Ziffer erreicht ist, in diesem Fall also die 7, und danach die nächsthöhere Stelle um eins erhöht und so weiter. Damit ist die Oktalzahl 10 identisch mit der Dezimalzahl 8.

Der Vorteil des oktalen Zahlensystems besteht in der leichten Umrechenbarkeit von Oktalzahlen in Dualzahlen und umgekehrt. Weil die Oktalzahlen „handlicher“ sind als ihre binären Äquivalente, werden sie gern zur Darstellung von Bytes verwendet.

0 = 000
1 = 001
2 = 010
3 = 011
4 = 100
5 = 101
6 = 110
7 = 111

Beispiel: $10111_8 = 4096 + 64 + 8 + 1 = 4169_{10}$

8^k	8^4	8^3	8^2	8^1	8^0
	4096	512	64	8	1

a_k	**1**	**0**	**1**	**1**	**1**
$a_k * 8^k$	4096	0	64	8	1

Hexadezimalzahlen

Es gibt in unserem Kulturkreis nur 10 gebräuchliche Ziffern. Aber es gibt eigentlich keinen Grund, auf Zahlensysteme mit mehr Ziffern zu verzichten. Das im Computerbereich gebräuchlich Hexadezimalsystem benutzt 16 Ziffern. Zusätzlich zu den Ziffern 0 bis 9 werden hier die Buchstaben A bis F als elfte bis sechzehnte Ziffer benutzt. Dieses Zahlensystem erlaubt auch eine relativ einfache Umrechnung von Dualzahlen in Hexadezimalzahlen und umgekehrt. Hierzu werden in der gleichen Weise wie bei den Oktalzahlen die Hexadezimalzahlen in vierstellige Dualzahlen und Gruppen zu je vier Dualzahlen zu Hexdezimalziffern umgewandelt. Ein Vorteil von Hexadezimalzahlen ist, dass sich ihre 16 verschiedenen Ziffern genau in einem Halbbyte darstellen lassen.

0 = 0000 8 = 1000
1 = 0001 9 = 1001
2 = 0010 A = 1010
3 = 0011 B = 1011
4 = 0100 C = 1100
5 = 0101 D = 1101
6 = 0110 E = 1110
7 = 0111 F = 1111

Beispiel: $10111_{16} = 65536 + 256 + 16 + 1 = 65809_{10}$

16^k	16^4	16^3	16^2	16^1	16^0
	65536	4096	256	16	1

a_k	**1**	**0**	**1**	**1**	**1**
$a_k * 16^k$	65536	0	256	16	1

3.2 Umrechnung einer Dezimalzahl in eine Dualzahl

Eine Dezimalzahl wird in eine Dualzahl umgewandelt, indem sie fortgesetzt durch 2 dividiert wird, wobei die Reste der Reihe die Dualziffern – beginnend mit der höchsten Stelle – bilden (Verfahren der fortgesetzten Division).

Beispiel: die Dezimalzahl 35 soll als Dualzahl geschrieben werden.

35 : 2 = 17 Rest **1**
17 : 2 = 8 Rest **1**
8 : 2 = 4 Rest **0**
4 : 2 = 2 Rest **0**
2 : 2 = 1 Rest **0**
1 : 2 = 0 Rest **1**

Die Dezimalzahl 35 lautet als Dualzahl: 100011.

3.3 Negative Dualzahlen

Zur Darstellung negativer Zahlen gibt es zwei Möglichkeiten:
a) mit Hilfe des Vorzeichens
b) mit Hilfe der Komplementbildung (Zweierkomplementbildung)

Vorzeichen

Das Vorzeichen ist eine binäre Variable. Es genügt deshalb zur Darstellung eine Dualziffer. Zweckmäßig wird folgende Festlegung getroffen:
0 bedeutet positiv
1 bedeutet negativ
Ansonsten bleibt die Dualzahl unverändert.

Beispiel an der Zahl 12:

Dualzahl					Dezimalzahl
Vorzeichen					
0	1	1	0	0	+ 12
1	1	1	0	0	- 12

Zweierkomplementdarstellung

Die Zweierkomplementdarstellung ist die gebräuchlichste interne Darstellung ganzer negativer

Zahlen. Auch Inside CPU verwendet sie. Das Zweierkomplement wird wie folgt berechnet:
- von der positiven Zahl wird das Einerkomplement durch Ersetzen von 0 durch 1 und 1 durch 0 gebildet,
- dann wird 1 hinzuaddiert.

Das Ergebnis ist das Zweierkomplement der vorgegebenen (negativen) Zahl.

Beispiel: Zweierkomplementdarstellung von –3 bei einer Wortlänge von 12 Bit

Die Zahl +3 entspricht im Dualsystem:	0000 0000 0011
Das Einerkomplement von +3 ist:	1111 1111 1100
Das Zweierkomplement entsteht durch Addition von + 1:	1111 1111 1101

Auch im Zweikomplement erkennt man negative Zahlen an einer 1 im höchstwertigen Bit. Sie wird aber hier nicht einfach der positiven Zahl als Vorzeichenbit vorangestellt. Die dem Zweierkomplement entsprechende positive Zahl ist nicht einfach zu erkennen. Ihr großer Vorteil besteht darin, dass mit ihr die Subtraktion auf die Addition zurückgeführt werden kann:

a – b $\Leftrightarrow$ a + Zweierkomplement (b)

3.4 Zahlenbereich von Inside CPU

Wenn Sie in Inside CPU Zahlenwerte als Daten abspeichern, müssen Sie die internen Grenzen des Arbeitsspeichers beachten. Es stehen maximal 12 Bit zur Verfügung. Da die Zahlen binär gespeichert werden, ergibt sich ein Umfang von 2^{12} = 4096 darstellbaren Zahlen. Bei Verzicht auf negative Zahlen würde sich der Zahlenraum erstrecken:

0 bis 4095

Zur Darstellung negativer Zahlen wird diejenige Hälfte der Zeichenkombinationen benötigt, die im höchstwertigen Bit eine 1 aufweisen, nämlich für die Zweierkomplemente. Der Zahlbereich erstreckt sich also:

von	-2048	100000000000
über	-1	111111111111
und	0	000000000000
und	+1	000000000001
bis	+2047	011111111111

Eine Eingabe von Zahlen außerhalb dieses Bereiches im Editierfenster ist nicht möglich. Gebrochene Zahlen (also Zahlen mit Kommastellen) sind ebenfalls nicht darstellbar. Achten Sie bei der Codierung arithmetischer Befehle darauf, dass das Rechenergebnis diesen Bereich nicht über- bzw. unterschreitet. Inside CPU gibt ggf. eine entsprechende Fehlermeldung aus.

3.5 Arithmetische Operationen mit Dualzahlen

Da die heute im Einsatz befindlichen Rechenanlagen zur Zahldarstellung fast ausschließlich das Dualsystem benutzen, werden im Rechenwerk die arithmetischen Operationen mit Dualzahlen durchgeführt.

Folgende arithmetische Grundoperationen können in Inside CPU-Programmen verwendet werden:
- Addition zweier Dualzahlen
- Multiplikation zweier Dualzahlen
- Subtraktion zweier Dualzahlen
- Division zweier Dualzahlen

3.5.1 Addition zweier Dualzahlen

Die Addition ist auf Grund des geringen Zeichenvorrates von nur zwei Ziffern höchst einfach. Es gelten folgende Additionsregeln:
0 plus 0 = 0
1 plus 0 = 1
0 plus 1 = 1
1 plus 1 = 10 (1 ist der Übertrag zur nächst höheren Stelle)

Nach dem obigen Schema können wir nun Bit für Bit die Addition zweier Dualzahlen vornehmen.

Beispiel: 317 + 369 = 686

Dual	Dezimal	
100111101	317	
+ 101110001	+ 369	
111 1	1	Übertrag
1010101110	686	

3.5.2 Multiplikation zweier Dualzahlen

Es gelten die folgenden Multiplikationsregeln:
0 mal 0 = 0
0 mal 1 = 0
1 mal 0 = 0
1 mal 1 = 1

Ähnlich wie im Dezimalsystem wird auch hier die Multiplikation auf Additionen mit entsprechenden Stellenverschiebungen zurückgeführt:

Beispiel: 13 * 6 = 78

Der Multiplikand sei 13 und der Multiplikator sei 6.

```
1101 * 110
   0000
  1101
 1101
  11          Übertrag
 1001110
```

3.5.3 Subtraktion zweier Dualzahlen

Bei der Subtraktion (Minuend minus Subtrahend) haben wir für eine Stelle der Differenz ein ähnliches Verhalten wie bei der Addition. Nur wird hierbei, wenn der Subtrahend größer ist als der Minuend an Stelle eines Übertrages zur nächsten Stelle ein „Borger“ (engl. Borrow) auftreten (wie wir es auch von der dezimalen Subtraktion aus der Schule her kennen).

Minuend	Subtrahend	Differenz	Borger
0	0	0	0
1	0	1	0
1	1	0	0
0	1	1	1

Beispiel: 369 – 317 = 52.

```
    Dual                Dezimal
  101110001               369
- 100111101             + 317
    1111                          Übertrag
  000110100                52
```

3.5.4 Division zweier Dualzahlen

Wie bei der Dezimalrechnung besteht das Rechenverfahren aus einer wiederholten Subtraktion des von Subtraktion zu Subtraktion nach rechts stellenverschobenen Divisors.

Beispiel: 65 : 5 = 13.

```
 1000001 : 101 = 1101
-101
 00110
 - 101
   0010
  - 0000
     101
   - 101
     0
```

Mögliche Divisionsreste werden in Inside CPU nicht berücksichtigt.

3.6 Codes

Da Computer zu einem großen Teil nicht nur zum Berechnen von Zahlen, sondern zum Bearbeiten von allgemeineren Daten benutzt werden, ist ein erheblich größerer Zeichenvorrat als der für Zahlensysteme notwendig. Diese Zeichenvielfalt muss dann allerdings z.B. den 8 Bits eines Bytes zugeordnet werden. Daher gibt es seit den Anfängen der EDV Tabellen, die den verschiedenen Bytes Buchstaben und andere Zeichen zuordnen.

Ausgangspunkt ist folgende Überlegung: wenn in einem Byte ohnehin nur Dezimalzahlen zwischen 0 und 255 bzw. zwischen –128 und +127 untergebracht werden können, dann kann man sich bei der Zeichendarstellung auch gleich darauf beschränken, in einem Byte nur die Dezimalziffern 0 bis 9 unterzubringen. 246 (= 256 – 10) Bytes stehen dann für andere Zeichen zur Verfügung. Wenn jede Stelle einer Dezimalzahl durch ein solches Byte dargestellt wird, spricht man von binär codierten Dezimalzahlen oder kurz: **BCD-Zahlen**, da sie lediglich die Ziffern dual darstellen, den Stellenwert jedoch aus dem Dezimalsystem übernehmen.

Beispiel: 94 als BCD-Zahl

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0

10^1	10^0
9	4

Dezimalziffern in einem ganzen Byte unterzubringen ist insofern verschwenderisch, als man dafür auch mit einem Halbbyte auskommen würde. Man spricht dann von der **gepackten Form**.

Beispiel: 94 als BCD-Zahl in gepackter Form

2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0
1	0	0	1	0	1	0	0

10^1	10^0
9	4

Jetzt wird auch verständlich, woher der erste prominente Code, der sog. **EBCDI-Code** seinen Namen bezog: der extended binary coded decimal interchange code war schlicht eine Erweiterung der binär codierten Dezimalzahlen um die Buchstaben des lateinischen Alphabets und einige Sonderzeichen, wie man sie für die ersten Programmiersprachen benötigte.

Während dieser Code noch mit deutlich weniger als 256 Zeichen auskam, wurde dieser Zeichenvorrat vom für die PC-Welt besonders wichtigen **sog. ASCII-Code** (American Standard Code for Information Interchange) voll genutzt, u.a. für besondere Buchstabenvarianten gängiger am lateinischen Alphabet orientierter Sprachen, für weitere Sonderzeichen und für semigraphische Zeichen zur besseren Gestaltung der zunächst zeichenorientierten Benutzungsoberflächen. Die ASCII-Tabelle ist dem Anhang zu entnehmen.

Da ein 8-Bit-Code wie ASCII nur 256 verschiedene Zeichen darstellen kann, ist er für die Aufnahme z.B. der vielen Schriftzeichen asiatischer Sprachen ungeeignet. Hierfür besteht aber ein

großer Bedarf, um Software einfacher in diese Sprachen übersetzen („lokalisieren“) zu können. Deshalb wurde der sog. **Unicode** als universeller 16-Bit-Code kreiert. Sie können einen Eindruck von diesem Code gewinnen, wenn Sie in Microsoft Word mit EINFÜGEN.SYMBOL das Symbolfenster öffnen und als Schriftart „Arial Unicode MS“ auswählen.

3.7 Programme

Ein Prozessor verarbeitet Eingabedaten mit Hilfe der gegebenen Befehle, die zusammen das Programm bilden, zu den gewünschten Ausgabedaten. Programme und Daten sind damit die entscheidenden Bestandteile der Datenverarbeitung.

Maschinenprogramme sind direkt in der Maschinensprache geschriebene Programme. Sie sind in der Regel sehr schnell und effizient. Ein großer Nachteil dieser Maschinenprogramme ist, dass sie prozessorspezifisch und damit nicht 1:1 auf Systeme mit anderen Prozessoren übertragbar sind. Nur innerhalb einer Prozessorfamilie sind sie kompatibel. Außerdem sind sie sehr schwer zu lesen, da sie nur aus Nullen (0) und Einsen (1) bestehen.

Jeder Computer beherrscht eine festgelegte Menge an – für sich sehr einfachen – Befehlen. Die Menge aller möglichen Befehle in ihren möglichen Abwandlungen nennt man die **Maschinensprache** eines Rechners. Maschinenprogramme bestehen nur aus diesen Maschinenbefehlen.

Der Prozessor versteht keine **Höheren Programmiersprachen** wie Cobol, Basic, C, C++ oder Java, sondern nur den Maschinencode. In diesen müssen die Anweisungen jeder Hochsprache übersetzt (kompiliert oder interpretiert) werden.

Der Maschinencode entsteht aus einer Folge von **Maschinenbefehlen**, die ihrerseits aus einem Operationsteil (Opcode) und einem Adressteil bestehen.

Code	Adresse	Befehl	Dezimal
0001	0000 1000	Lade	8

Der **Operationsteil** enthält den binär verschlüsselten Code des auszuführenden Befehls. Er gibt an, was zu tun ist (z.B. Lade, Addiere). Der **Adressteil** enthält die Adresse der Speicherstelle, mit deren Inhalt die Operation durchgeführt werden soll. Bei einer **Einadressmaschine** enthält der Adressteil nur eine Adresse. Bei einer **Mehradressmaschine** enthält der Adressteil mehrere Adressen.

Die Menge aller für einen Prozessor definierten Befehle wird als sein **Befehlsvorrat** oder **Befehlssatz** bezeichnet. Sie wird durch den Aufbau der CPU, insbesondere des Steuerwerkes (Befehlsdecodierung) und der ALU, festgelegt. Nach der Zweckbestimmung kann man folgende Gruppen (Befehlstypen) unterteilen:

Verarbeitungsbefehle

Verarbeitungsbefehle steuern die ALU. Sie bewirken, dass ein oder zwei Operanden zu einem Ergebnis verarbeitet werden. Diese Befehle müssen die Art der Operation enthalten. Beispiele für Operationen sind die Addition, Subtraktion, Multiplikation und Division.

Transportbefehle

Transportbefehle dienen der Bewegung von Daten zwischen den Speicherzellen. Dabei handelt es sich bei den meisten Rechnern um eine Register- und eine Speicherzelle. Beispiele für Transportbefehle sind Laden und Speichern.

Sprungbefehle

Sprungbefehle steuern die Auswahl des nächsten Befehls. Man unterscheidet:

- **Unbedingte Sprünge** zu einer Zieladresse (Inside CPU: Springe)
- **Bedingte Sprünge**, bei denen nur dann ein Sprung zu einer Zieladresse durchgeführt wird, wenn im Flagregister die Erfüllung bestimmter Bedingungen angezeigt, ansonsten wird das Programm in normaler Reihenfolge weitergeführt (Inside CPU: Springe, wenn negativ; Springe, wenn 0)
- **Sprünge mit Rückkehrabsicht**, wenn ein Unterprogramm aufgerufen wird (Inside CPU: Springe in Unterprogramm). Dieser Sprung bereitet zumindest vor, dass man den aktuellen Stand der bisherigen Programmabarbeitung sichert und mit Hilfe eines Rückkehrsprunges (Inside CPU: Springe zurück) zurückkehren kann.

Steuerbefehle

Steuerbefehle wirken sich nicht auf die Daten aus. Erforderlich ist z.B. ein „Halt“-Befehl (Inside CPU: Stop), der das Ende des Programms anzeigt.

3.8 Von-Neumann-Architektur

John von Neumann wurde am 28.12.1903 in Budapest geboren. Er war amerikanischer Mathematiker österreichisch-ungarischer Herkunft. Zusammen mit Oskar Morgenstern begründete er die Spieltheorie, die im engen Zusammenhang mit den Gesetzen des Wirtschaftsablaufs steht. Bereits 1946 definierte Neumann seine theoretischen Forderungen an die Konstruktion einer allgemeinen einsetzbaren Rechenanlage. Er beschrieb den Aufbau und die Funktion eines Rechners für die elektronische Datenverarbeitung. Deshalb wird ein klassischer Computer als Von-Neumann-Rechner und seine Architektur als Von-Neumann-Architektur bezeichnet.

Die Von-Neumann-Prinzipien

Nachfolgend sollen die wesentlichen Prinzipien der klassischen Von-Neumann-Rechnerarchitektur noch einmal zusammengefasst werden:

1. Der Rechner besteht aus fünf Funktionseinheiten:
- Steuerwerk oder Leitwerk
- Rechenwerk
- Speicher
- Eingabewerk
- Ausgabewerk

2. Die Struktur des Von-Neumann-Rechners ist unabhängig von den zu bearbeitenden Problemen. Zur Lösung eines Problems muss von außen eine Bearbeitungsvorschrift, das Programm, eingegeben und im Speicher abgelegt werden. Ohne dieses Programm ist die Maschine nicht arbeitsfähig. Programme, Daten, Zwischen- und Endergebnisse werden in demselben Speicher abgelegt.

3. Der Speicher ist in gleichgroße Speicherzellen unterteilt, die fortlaufend nummeriert sind. Über die Nummer (Adresse) einer Speicherzelle kann deren Inhalt abgerufen oder verändert werden.

4. Aufeinanderfolgende Befehle eines Programms werden in aufeinanderfolgende Speicherzellen abgelegt. Das Ansprechen des nächsten Befehls geschieht vom Steuerwerk aus durch Erhöhen der Befehlsadresse um 1.

5. Durch Sprungbefehle kann von der Bearbeitung der Befehle in der gespeicherten Reihenfolge abgewichen werden.

6. Es gibt zumindest
   - arithmetische Befehle wie Addieren, Subtrahieren, Multiplizieren, Dividieren etc.
   - logische Befehle wie Vergleichen etc.
   - Transportbefehle Laden und Speichern etc.
   - sonstige Befehle wie Schieben, Unterbrechen, Warten usw.

7. Alle Daten (Befehle, Adressen usw.) werden binär codiert. Geeignete Schaltwerke im Steuerwerk und an anderen Stellen sorgen für die richtige Entschlüsselung.

Die Architektur des Von-Neumann-Rechner fällt in die Klasse der sogenannten SISD-Architekturen (single instruction stream, single data). Diese Architekturen sind gekennzeichnet durch einen Prozessor (Ein-Prozessor-System) bestehend aus Steuer- und Rechenwerk sowie die Erzeugung einer Befehls- und Operandenfolge mit streng sequentieller Abarbeitung.

3.8.1 Steuerwerk

Das Leitwerk (engl.: control unit), das auch häufig als Steuerwerk bezeichnet wird, sorgt für die Durchführung der einzelnen Befehle eines Programms. Es steuert den Ablauf des Befehls- und Datenflusses und bestimmt mit seinem Taktgeber die Programmablaufgeschwindigkeit:
- Laden der Befehle (Anweisungen des gerade bearbeiteten Programms) aus dem Speicher in der richtigen Reihenfolge
- Decodieren der Befehle
- Interpretation der Befehle
- Versorgung der an der Ausführung der Befehle beteiligten Funktionseinheiten mit den nötigen Steuersignalen (mittels Steuerbus).

3.8.2 Rechenwerk

Die Verarbeitung von Daten findet im Rechenwerk (ALU und Akkumulator) statt. Die zu verarbeitenden Daten nennt man Operanden oder Argumente. Zur Verarbeitung müssen die Daten kurz im Rechenwerk gespeichert werden. Das Rechenwerk besitzt zu diesem Zweck schnelle

Speicher, sogenannte Register (hier: Akkumulator und Operandenregister). Im Rechenwerk werden überwiegend Binärworte verarbeitet. Die Operationen Addition, Subtraktion, Multiplikation und Division können als Grundoperationen eines Rechenwerkes angesehen werden.

Weitere Ausführungen zur ALU finden sie im Abschnitt 5.3.

3.8.3 Hauptspeicher

Im Arbeitsspeicher eines Computers werden die Fächer als Speicherzellen bezeichnet. Die Nummerierung der Fächer entspricht den sogenannten Adressen der einzelnen Speicherzellen. Der Hauptspeicher (oder Arbeitsspeicher) ist das Kurzzeitgedächtnis eines Rechners. Er speichert Daten in binärer Form, die im Rechenwerk verarbeitet werden sollen. Weiterhin werden auch die einzelnen Programmbefehle in binärer Form in diesem Speicher festgehalten.

Die Speicherzellen liegen in einer Matrix angeordnet vor. Die Zeilen dieser Matrix nennt man Speicherwort bzw. Wort . Gängige Größen für Speicherworte sind: 1 Bit, 4 Bit (1 Nipple), 8 Bit (1 Byte), 12 Bit, 16 Bit (1 Wort), 32 Bit (1 Doppelwort), 64 Bit, 128 Bit.

Will man Daten aus dem Hauptspeicher lesen oder hineinschreiben, dann muss man zunächst die Speicherstelle angeben, auf die man zugreifen will. Danach kann man das zugehörige Speicherwort über den Datenbus lesen bzw. schreiben.

3.9 Befehlszyklus

Unter dem Befehlszyklus (Operationszyklus, Von-Neumann-Zyklus) versteht man die Abfolge der Prozessorzustände, die bis zum Abruf des nächsten Befehls aus dem Arbeitsspeicher durchlaufen werden.

Im ersten Schritt wird ein Befehl (oder auch ein Teil davon) aus der Befehlsschlange in das Instruktionsregister geladen (fetch). Dabei wird in echten Prozessoren vorher abgefragt, ob ein Interrupt-Bit gesetzt ist und ggf. ein Interrupt auszuführen ist. In der Zwischenzeit wird die Befehlsschlange durch eine andere Systemkomponente wieder mit Befehlen aufgefüllt.

In der Inside CPU besteht dieser erste Schritt (**fetch**) aus drei Phasen:
- Bereitstellen der Befehlsadresse aus dem Befehlszähler im Adressregister und Erhöhen des Befehlszählers um 1
- Inhalt der Befehlsadresse im Datenregister bereitstellen (Lesezyklus)
- Befehlstransport in das Instruktionsregister und Decodieren

Im zweiten Schritt (auch Exekutionszyklus oder **execute** genannt) wird der geladene Befehl vom Steuerwerk ausgeführt. Dabei hängt die Anzahl der Teiloperationen vom jeweiligen Befehl ab. Es werden Eingaben, Ausgaben, Lese- oder Schreibvorgänge, interne Rechenoperation und Datenübertragungen durchgeführt.

Inside CPU führt die Teilschritte je nach Befehlsart unterschiedlich aus. In vielen Fällen werden mit dem Adressteil des Befehls Schreib- oder Lesevorgänge ausgeführt, Rechenoperationen oder Datentransporte schließen sich an. Jeder Einzelschritt wird in der Statuszeile dokumentiert.

Ein Befehlszyklus endet mit der vollständigen Abarbeitung aller Teilschritte des aktuellen Befehls.

Die Datenverbindung vom Prozessor zum Arbeitsspeicher und umgekehrt wird durch den Schreib- bzw. Lesezyklus realisiert. Beim **Schreibzyklus** (z.B. Speichern) steht die abzuspeichernde Information im Datenregister, die Arbeitsspeicheradresse als Zielort steht im Adressregister. Das Steuerwerk löst nun aufgrund des aktuellen Befehls, der den Schreibvorgang enthält, einen Schreibimpuls aus, der den Speicher aktiviert und die anliegenden Datenbits an der anliegenden Adresse ablegt. Je nach Geschwindigkeit des Speichermediums muss ein echter Prozessor (oder der Speicher) Wartezustände zur Synchronisation durchlaufen.

Beim **Lesezyklus** steht die Leseadresse im Adressregister. Durch einen Leseimpuls des Prozessors gelangt der Inhalt der angegebenen Adresse in das Datenregister und steht dem Prozessor zur Verfügung.

Inside CPU untermalt den Schreib- bzw. Leseimpuls durch ein Geräusch. Die farbige Fließbewegung im Adress- und Datenbus zeigt die Richtung des Adress- und Datenflusses an.

Ein **Interrupt** als Unterbrechung eines laufenden Programms kann in Inside CPU nur symbolisch als Hardwareinterrupt (durch Klicken mit der Maus auf die Schaltflächen „Reset“, „Editor“ oder „Hauptspeicheranalyse“) von außen ausgelöst werden. Dabei wird im Interruptregister eine Adresse erzeugt, die nicht im Arbeitsspeicher liegt. Da Inside CPU keine Bildschirmein- und -ausgaben, Diskettenspeicherzugriffe und höhere Verwaltungsfunktionen (Betriebsystemaufgaben) simuliert, die auf Interruptaufrufen basieren, ist eine ausführlichere Interruptbehandlung nicht notwendig.

4 Die Fenster von Inside CPU

Inside CPU ist sehr übersichtlich. Das Lernprogramm kommt mit drei Fenstern aus: im Editierfenster können Sie Programme visuell programmieren, deren Abarbeitung Sie anschließend im Simulationsfenster dynamisch verfolgen können. Das Hauptspeicheranalysefenster ist ein Hilfsfenster zum Simulationsfenster, um nähere Informationen zum jeweiligen Zustand des Hauptspeichers erhalten zu können.

4.1 Das Editierfenster

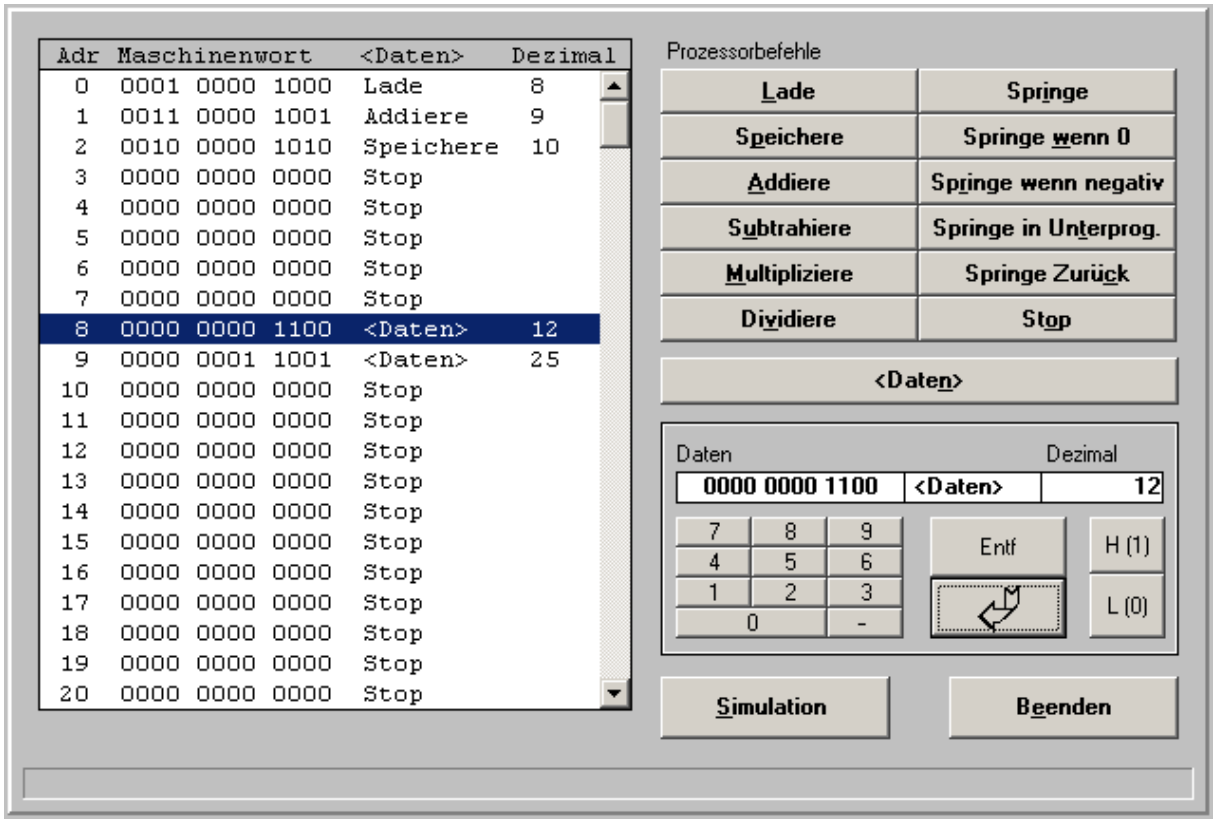
Der Editierfenster dient zum Laden und Modifizieren vorhandener bzw. zum Erstellen neuer Programme, die im Arbeitsspeicher abgelegt werden. Alle Inside CPU-Programme werden mit der Extension ".sim" abgespeichert und dürfen nicht mit anderen Editoren als dem von Inside CPU erstellt oder verändert werden!

Auf der rechten Seite des Bildschirms sind alle Inside CPU-Befehle (Lade, Speichere, Addiere, Subtrahiere, Multipliziere, Dividiere usw.) aufgeführt, die in Inside CPU-Programmen verwendet werden können.

Bei den meisten Befehlen werden Sie noch zur Eingabe einer Adresse (Platznummer) bzw. eines Datenwertes aufgefordert. Hierfür ist der einem Taschenrechner nachempfundene Ziffernblock vorgesehen. Mit der <Enter>-Taste wird der Befehl bzw. der Datenwert in den Arbeitsspeicher von Inside CPU auf der linken Bildschirmseite übernommen.

Der Arbeitsspeicher wird durch vier Spalten symbolisiert:

- die erste Spalte enthält die aktuelle dezimale **Arbeitsspeicheradresse** (Platznummer),
- die zweite Spalte enthält bei Befehlen den **Operationsteil** und den **Adressteil** bzw. bei Datenfeldern den **Datenwert** in dualer Schreibweise,
- die dritte Spalte nimmt bei Befehlen den Operationsteil als Text (z.B. Addiere, Dividiere), bei Datenfeldern den Hinweis <Daten> auf,
- darauf folgt in der vierten (letzten) Spalte bei Befehlen der Adressteil bzw. bei Datenfeldern der Datenwert in dezimaler Schreibweise.



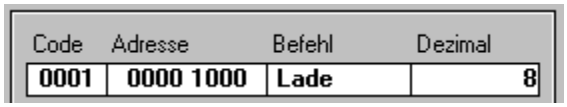
Ein „blauer Balken“ kennzeichnet den aktuellen Arbeitsspeicherplatz, auf den man jeweils zugreifen kann. Er steht zu Beginn immer auf dem Speicherplatz mit der Adresse 0. Ist der jeweilige Speicherplatz mit einem Befehl oder mit Daten beschrieben worden, springt der „blaue Balken“ automatisch zur nächsten Adresse. Er lässt sich jedoch auch mit Hilfe der Maus bewegen.

Im Editierfenster kann das Programm nur einem sog. **Trockentest** unterzogen werden, ohne es zu starten. Deshalb lassen sich auch noch keine Programmergebnisse einsehen, die erst im Laufmodus erzeugt und danach in der Regel durch Speicherbefehle zusätzlich in den Arbeitsspeicher geschrieben werden.

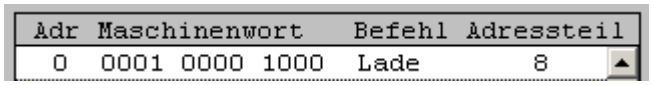
Befehlseingabe

Man betätigt zunächst die Schaltfläche für den gewünschten Befehl (Operationsteil) und gibt anschließend auf dem Ziffernblock des Editierfensters die gewünschte Adresse (Adressteil) ein, am einfachsten mit der Maus. Die Adresse darf nicht größer als 255 sein.

Beispiel: Lade 8



In den vier weißen Anzeigefeldern oberhalb des Ziffernblocks sieht man sofort, ob der Befehl korrekt editiert wurde. Mit der <Enter>-Taste wird der binär codierte Befehl in die aktuelle Arbeitsspeicherzelle übernommen.

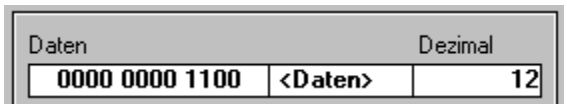


Alle Speicherplätze sind mit dem Stopbefehl vorbelegt. Daher ist es nicht notwendig, am Ende eines Programms noch einmal den Stopbefehl einzugeben.

Dateneingabe

Daten werden an der gewünschten Arbeitsspeicherstelle mit der <Daten>-Taste sowie dem Ziffernblock eingegeben. Die Datenwerte können entweder mit der Zehner-Tastatur dezimal oder mit der High- und der Low-Taste [H (1) und L (0)] dual eingegeben werden. Die Daten dürfen zwischen -2048_{10} und $+2047_{10}$ liegen.

Beispiel: Datenwert 12



In den weißen Anzeigefeldern oberhalb des Ziffernblocks erscheint der Datenwert links dual und rechts dezimal. Mit der <Enter>-Taste wird die Dualzahl in die aktuelle Arbeitsspeicherzelle übernommen.

Eingabekorrektur

Bereits gefüllte Speicherplätze können überschrieben werden. Der zuletzt eingegebene Befehl bzw. Datenwert kann im Menü BEARBEITEN mit RÜCKGÄNGIG: EINGABE ÜBERNEHMEN wieder zurückgenommen werden.

Insbesondere wenn man Daten in zu niedrigen Speicherstellen abgelegt hat, weil man die Zahl der notwendigen Programmbefehle unterschätzt hat, ist es hilfreich, mit dem Befehl ZEILEN EINFÜGEN aus dem Menü BEARBEITEN zusätzlichen Platz für Befehle zu schaffen. Umgekehrt können mit dem Befehl ZEILEN LÖSCHEN Zeilen auch wieder entfernt werden.

Sichern und Laden von Inside CPU-Programmen

Nur aus dem Editierfenster heraus können Sie mit den Standarddialogfeldern von Windows neu erstellte oder zwischenzeitlich bearbeitete Inside CPU-Programme auf der Festplatte sichern bzw. abgespeicherte Programme laden. Inside CPU-Programme werden automatisch mit der Dateiendung .sim gespeichert. Ist unter dem eingegebenen Namen bereits eine Datei abgespeichert, erscheint die Meldung: „*Datei* besteht bereits. Möchten Sie sie ersetzen?" Das bisher unter diesem Dateinamen gespeicherte Programm geht dann verloren!

Wechsel zum Simulationsfenster

Um Ihr fertiges oder auch nur vorläufiges Programm zu testen, klicken Sie auf die Schaltfläche „Simulation“.

Beenden von Inside CPU

Mit der Schaltfläche „Beenden“ oder mit dem Befehl BEENDEN aus dem Menü DATEI wird Inside CPU verlassen.

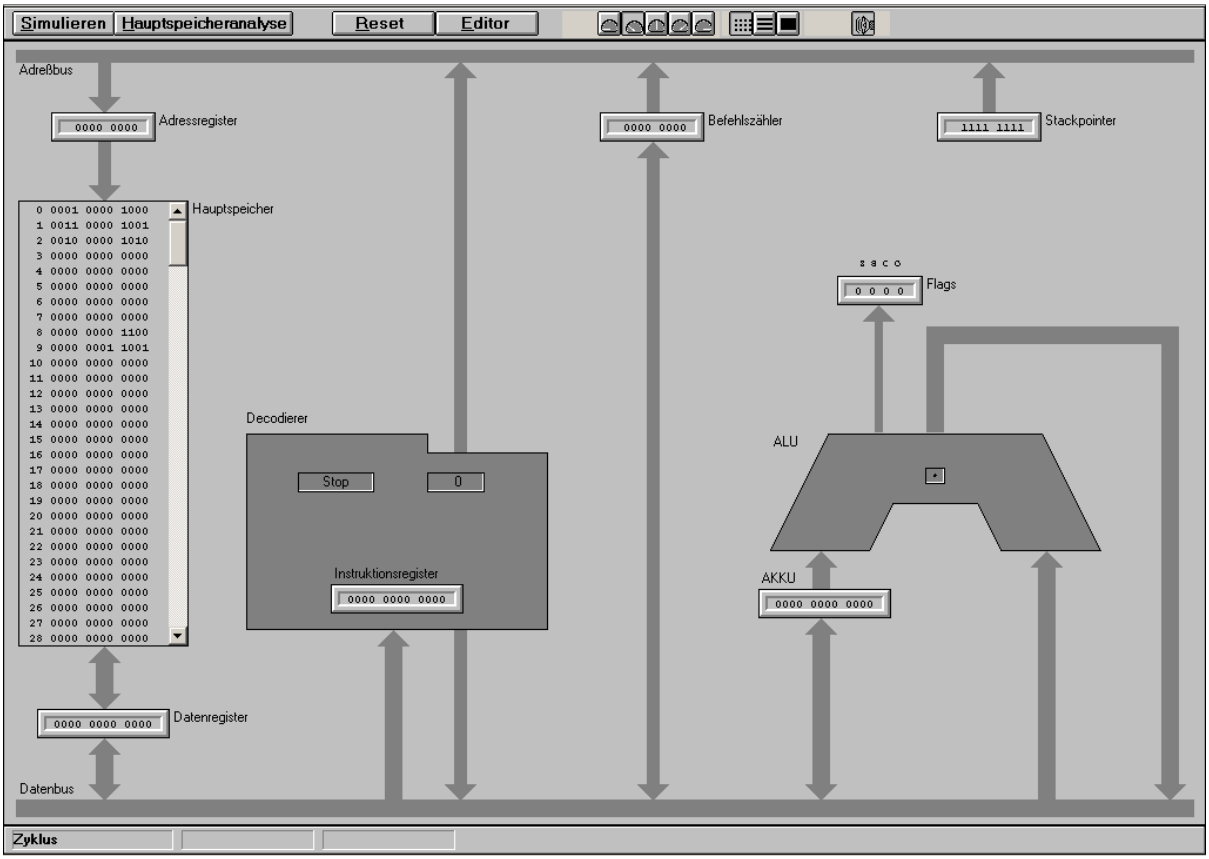
4.2 Das Simulationsfenster

Im Simulationsfenster werden die typischen Vorgänge in einer CPU bei der Abarbeitung von Befehlen bzw. Programmen veranschaulicht. Die wichtigsten Bestandteile einer CPU (Adressbus, Adressregister, Akkumulator (AKKU), ALU, Befehlszähler, Datenbus, Datenregister, Decodierer, Flagregister, Hauptspeicher, Instruktionsregister, Stackpointer) sind dargestellt und mit den entsprechenden Bezeichnungen versehen. Die Lage der einzelnen Bestandteile und ihrer Verbindungswege hat mit der tatsächlichen Anordnung in einer echten CPU wenig zu tun und wurde nach didaktischen und optischen Gesichtspunkten ausgewählt. Außerdem wurden aus Platzgründen und zur besseren Übersicht einige Vorgänge vereinfacht dargestellt. Hierzu zählen insbesondere Abläufe im Rechenwerk und der Zugriff auf den Arbeitsspeicher.

Die linke Seite des Bildschirms wird von der graphischen Darstellung des Hauptspeichers (Arbeitsspeichers) eingenommen, wie sie aus dem Editierfenster vertraut ist. Die beiden im Editierfenster rechts stehenden Spalten Befehl und Adressteil, die nur zur besseren Lesbarkeit der Befehle hinzugefügt wurden, werden im Simulationsfenster aus Platzgründen nicht dargestellt. Wechseln Sie zur Probe einmal durch ein einfaches Klicken auf die Schaltfläche „Editor“ zwischen dem Simulationsfenster und Editierfenster hin und her! In den verbleibenden Spalten stehen links die zweistelligen dezimalen Arbeitsspeicheradressen und rechts die binär codierten Befehle, die sich aus dem Operationsteil (vier Stellen links) und dem Adressteil (acht Stellen rechts) zusammensetzen, bzw. zwölfstellige Dualzahlen.

Mit den fünf Schaltflächen, auf denen ein Tachometer abgebildet ist, kann man das Tempo festlegen, in dem Inside CPU-Programm ablaufen sollen. Die Simulationsgeschwindigkeit kann fünfstufig variiert werden. Mit den drei Schaltflächen rechts daneben kann man die Simulationsabschnitte festlegen. Sie lassen sich von Einzelschritten (einzelner Zyklus) eines Befehls über einen kompletten Befehlszyklus (einzelner Befehl) bis hin zum Gesamtprogramm (Gesamtprogramm) erweitern. Zu Beginn sind Geschwindigkeit und Laufmodus auf "Langsam" und "einzel-

ner Zyklus" eingestellt. Mit der Lautsprecher-Schaltfläche kann die akustische Untermalung der Simulation ein- und ausgeschaltet werden.



Die Statuszeile am unteren Bildschirmteil zeigt ganz links an, in welcher Phase der Abarbeitung des jeweiligen Befehls sich das Programm gerade befindet (Befehlszyklus bzw. Von-Neumann-Zyklus). Die Statusfelder rechts daneben liefern ergänzende Informationen.

Mit der Schaltfläche „Simulieren“ startet man die Simulation. Während des Programmablaufs kann dieser mit der Schaltfläche „Stop“ unterbrochen und mit „Simulieren“ wieder fortgesetzt werden. Vom Laufmodus aus kann jederzeit mit der Schaltfläche „Hauptspeicheranalyse“ das Hauptspeicheranalysefenster aufgerufen werden und von diesem wieder in das Simulationsfenster zurückgegangen werden. Auch hierdurch wird der Programmablauf nur unterbrochen und nicht „von vorne“ neu gestartet.

Genau dies bewirkt die Schaltfläche "Reset" (Re-Initialisierung der simulierten CPU): das Programm im Arbeitsspeicher bleibt erhalten, aber der Befehlszähler wird auf Null gesetzt, d.h., der Programmablauf beginnt wieder mit der Anfangsadresse 00. Die CPU-Graphik wird gleichfalls in den Anfangszustand zurückversetzt.

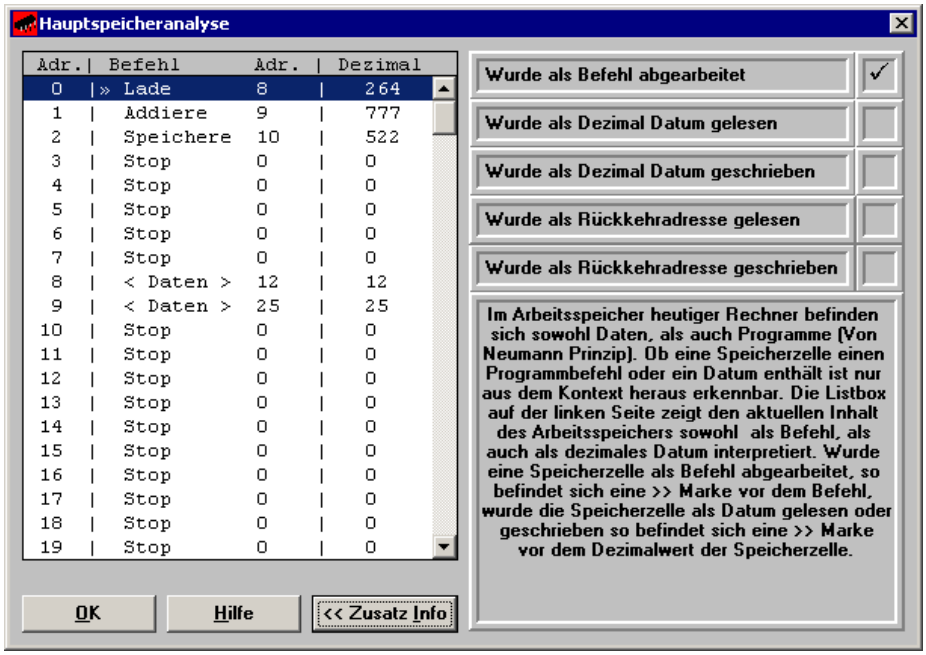
4.3 Das Hauptspeicheranalysefenster

Das Hauptspeicheranalysefenster zeigt detailliert das im Arbeitsspeicher abgelegte Programm. Man das Programm und seinen Abarbeitungsstand überprüfen und Programmergebnisse ein-

sehen, die durch Speicherbefehle zusätzlich in den Arbeitsspeicher geschrieben wurden.

Beim Öffnen des Hauptspeicheranalysefensters werden zunächst nur die vier Spalten des Arbeitsspeichers angezeigt:

- Die erste Spalte enthält die dezimalen Arbeitsspeicheradressen.
- Die zweite Spalte enthält bei Befehlen den Operationsteil als Text, bei Datenfeldern den Hinweis <Daten>.
- Die dritte Spalte enthält bei Befehlen die Adresse in dezimaler Schreibweise, an der die zu verarbeitenden Daten stehen, und bei positivem Datenwert den Datenwert bzw. bei negativem Datenwert den aus dem negativen Datenwert ausgerechneten Adresswert.
- Die vierte Spalte enthält bei Befehlen den kompletten Maschinenbefehl (Operationsteil plus Adressteil) bzw. bei Datenfeldern der Datenwert in dezimaler Schreibweise.



Die Zeichen >> kennzeichnen die Arbeitsspeicherplätze, die schon abgearbeitet sind. Der blaue Balken steht immer auf dem Speicherplatz mit der Adresse 0. Er kann mit der Maus verschoben werden. Mit der Schaltfläche „Zusatz Info >>“ erhält man zu jedem Speicherplatz nähere Angaben,

- ob er als Befehl abgearbeitet wurde (Wurde als Befehl abgearbeitet),
- ob er als Datenwert aus dem Arbeitsspeicher geholt wurde (Wurde als Dezimal Datum gelesen) bzw. im Arbeitsspeicher gespeichert wurde (Wurde als Dezimal Datum geschrieben),
- ob er bei einem Sprung in ein Unterprogramm als Rückkehradresse gelesen bzw. geschrieben wurde (Wurde als Rückkehradresse gelesen, Wurde als Rückkehradresse geschrieben).

5 Die Funktionselemente von Inside CPU

Der Prozessor eines Rechners wird auch CPU (Central Processing Unit) genannt. Teile des Prozessors sind das Rechen- und Steuerwerk sowie die Steuereinheit. Das Rechen- und das Steuerwerk führt alle Berechnungen durch. Die Steuereinheit ist zuständig für den Datenaustausch zwischen Rechen- und Steuerwerk, Arbeitsspeicher und anderen Systemkomponenten.

Die Bestandteile eines Mikroprozessors und die Elemente seiner unmittelbaren Umgebung werden in Inside CPU im Simulationsfenster graphisch dargestellt (vgl. Abschnitt 4.2). Grundsätzliche Elemente eines Von-Neumann-Rechners haben Sie bereits im Abschnitt 3.8 kennen gelernt. Hier werden die übrigen, im Simulationsfenster dargestellten Komponenten erläutert.

5.1 Arbeitsspeicher und Stapelspeicher (Stack)

Der Arbeitsspeicher von Inside CPU besteht aus 256 Speicherstellen mit einer Wortgröße von 12 Bit und enthält somit 256 * 12 Bit = 3072 binäre Speicherzellen. In jeder Speicherstelle lässt sich entweder ein Maschinenbefehl oder ein Dezimaldatum darstellen. Ein Befehl besteht grundsätzlich aus einem Operationsteil und einem Adressteil. Mit einer Adressbusbreite von 8 Bit sind 256 (= 2^8) Speicherwörter adressierbar, mit den verbleibenden 4 Bits lassen sich 16 verschiedene Befehle codieren. Damit kann ein Speicherwort einen Befehl und eine Adresse aufnehmen. Inside CPU modelliert also eine Ein-Adress-Maschine. Vor dem jeweiligen Speicherfeld steht zur besseren Orientierung noch einmal die Adresse in dezimaler Form, durchnummeriert von 0 bis 255.

Die interne Aufteilung eines solchen Feldes hängt davon ab, ob reine Daten oder Befehle in ein solches Feld abgespeichert werden. Sollen nur Daten gespeichert werden, so stehen alle 12 Bits zur Verfügung. Hieraus ergibt sich, dass nur Zahlen speicherbar sind, die im Intervall -2048 bis +2047 liegen: 2^{12} = 4096. Das höchstwertigste Bit fungiert als Vorzeichenbit. Daher ist insbesondere bei der Codierung arithmetischer Befehle darauf zu achten, dass das Rechenergebnis diesen Bereich nicht über- bzw. unterschreitet. Das Programm gibt ggf. eine entsprechende Fehlermeldung aus.

Im Rahmen der Unterprogrammtechnik ist ein Speicherbereich des Arbeitsspeichers von Bedeutung, der als Stapelspeicher bezeichnet wird. Der Stapelspeicher (engl. Stack) ist ein Stoß von Elementen, ganz ähnlich einem Stapel von Blättern auf einem Schreibtisch. Der Name „Stapelspeicher“ deutet bereits seine Eigenschaften an:

- Er stapelt Daten in der Reihenfolge wie sie hereinkommen und gibt sie in der umgekehrten Reihenfolge wieder heraus. Man nennt diese Art des Abspeicherns auch last in – first out.
- Jeder Aufruf eines Unterprogramms erhält ein zugewiesenes Stück des Stapelspeichers (Teil des Arbeitsspeicher).

Er ist mit der Adresse des letzten Speicherplatzes ("11111111" = 255) vorbelegt. Diese wird bei jedem weiteren Unterprogrammaufruf um 1 verkleinert, beim Rücksprung wieder um 1 vergrößert.

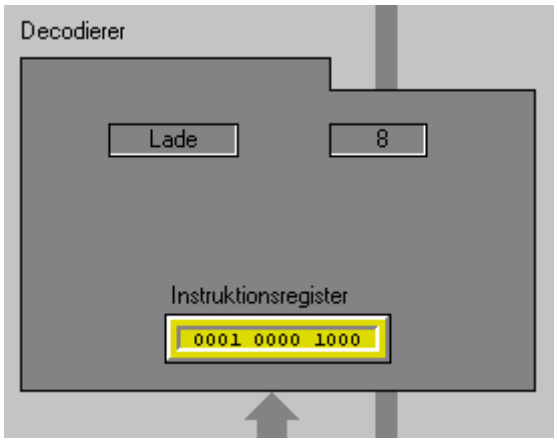
Ein Stapelspeicher (Stack) gestattet den Zugriff immer nur auf das oberste Element. Ein neues Element kann also immer nur oben auf dem Stapel abgelegt werden und umgekehrt kann immer nur das oberste Element heruntergenommen werden. Wegen dieser Eigenschaft wird er auch als LIFO-Datenstruktur (last in, first out) bezeichnet.

Die beiden wichtigsten Operationen (Element ablegen und Element entnehmen) eines Stapelzeigers (stacks) werden PUSH und POP genannt. PUSH speichert einen Wert in eine leere Speicherzelle. Der Stapelzeiger ist danach auf diese Zelle gerichtet. Wird ein neuer Wert gespeichert, so wächst der Stapelzeiger von unten nach oben (wie bei den Blättern auf einem Schreibtisch). POP gibt den zuletzt eingegebenen Wert zurück und setzt den Stapelzeiger auf den vorletzten Wert.

5.2 Decodierer

Alle Befehle, die eine CPU ausführen kann, werden durch einen sog. Befehls- oder Operationscode eindeutig definiert. Die spezifischen Bitfolgen werden vom jeweiligen Prozessorhersteller festgelegt. In der Regel stehen Hilfsprogramme - z.B. Assembler - zur Verfügung, die diese Bitfolgen - ähnlich wie in Inside CPU - aus mehr oder weniger selbsterklärenden Kurzwörtern (mnemotechnische Befehle wie "MOV", "ADD", "JMP" ..) erzeugen.

Der Befehlscode spaltet sich in einen Operationsteil (Opcode) und einen (oder mehrere) Adressteil(e) auf. In Inside CPU wird nur ein Adressteil verwendet (Ein-Adress-Maschine). Der Befehlscode hat eine Länge von 12 Bit. Die ersten 4 Bits (erste Tetrade) sind der Operationsteil, die nächsten 8 Bit (zweite und dritte Tetrade) sind der Adressteil. Mit den 4 Operations-Bits lassen sich insgesamt 16 verschiedene Befehle verschlüsseln und mit den 8 Adress-Bits 256 Speicherplätze adressieren.



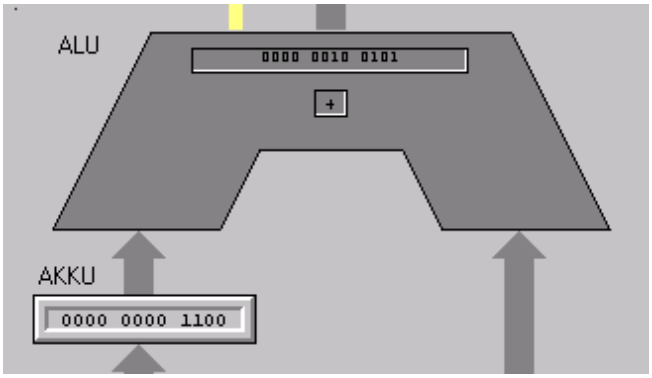
Der Decodierer entschlüsselt den im Instruktionsregister zwischengespeicherten Befehlscode, indem er ihn zunächst in Operations- und Adressteil aufspaltet (in PC-Prozessoren geschieht das durch das Lesen aufeinanderfolgender Bytes). Dann decodiert er den Operationsteil, um festzustellen, welcher Befehl auszuführen ist. Dieser Befehl kann nun in Verbindung mit der zugehörigen Adresse durch das Steuerwerk der CPU abgearbeitet werden.

5.3 Rechenwerk oder ALU (arithmetic logic unit)

ALU ist die gebräuchliche Abkürzung für arithmetic logic unit - das Rechenwerk im engeren Sinne. Hier werden arithmetische und logische Operationen (Vergleichen, Verschieben, Vorzeichenbestimmung Umformen, Runden) durchgeführt. Es besteht im Wesentlichen aus Registern und Addierschaltungen, weil die meisten Rechenoperationen auf elementare Additionen zurückgeführt werden können. Ergebnisse werden im Akkumulator, dem wichtigsten Register des Rechenwerks, zwischengespeichert. Bei vielen PC-Prozessoren gibt es noch eine Reihe weiterer Register für zusätzliche Operanden, z.B. für den Übertrag, für das stellenweise Verschieben, für die Komplementbildung (binäre Subtraktion) usw.

Es würde aber den Rahmen dieses CBT-Programms sprengen, die Gesamtheit dieser Register und deren komplexes Zusammenspiel darzustellen. Das Rechenwerk von der Inside CPU besteht deshalb nur aus zwei Registern:

- dem Akkumulator, der den ersten Operanden und später das Ergebnis der Rechnung enthält,
- und einem Operandenregister, das den zweiten Operanden aufnimmt.



Die Rechenoperation wird durch das Rechenzeichen in der ALU angezeigt, den binären Wert des Akkumulators können Sie im Operandenregister oberhalb des Rechenzeichens ablesen.

5.4 Register

Das Register ist eine Funktionseinheit zum Speichern eines Binärwortes. Jeder Computer besitzt mehrere Register (hier: Adressregister, Akkumulator, Befehlszähler, Datenregister, Flagregister, Instruktionsregister, Stackpointer). Ein Register kann ein Binärwort für beliebig lange Zeit speichern. Dieses Binärwort darf maximal aus n Bits bestehen. Man sagt dann, das Register habe die Breite n.

5.4.1 Adressregister

Das Adressregister ist ein Speicherplatz (Register), das die Adresse speichert.

5.4.2 Akkumulator (AKKU)

Einer der Eingänge der ALU verfügt über ein besonderes Register, den Akkumulator. Der Akkumulator ist das einzige Allzweckregister. Er ist als ein Operand an jeder Berechnung (z.B. Subtraktion) beteiligt und nimmt danach immer das Ergebnis auf. Daher der Name: die Ergebnisse aufeinanderfolgender Berechnungen werden hier akkumuliert und bilden schließlich das Endergebnis.

Benötigt wird der Akkumulator immer dann, wenn eine der vier in diesem Programm möglichen Grundrechenarten (Addition, Subtraktion, Multiplikation, Division) ausgeführt wird oder ein Speicherinhalt des Arbeitsspeichers zu verschieben bzw. neu zu füllen ist.

Zur Ausführung einer Grundrechenart benötigt man immer 2 Zahlen (Operanden) und eine Rechenanweisung (arithmetischen Operator). Im Akkumulator steht grundsätzlich der 1. Operand oder - anders ausgedrückt - der Operand links vom Operatorenzeichen. Der 2. Operand wird dann zum ersten addiert, vom ersten subtrahiert usw.. Das Ergebnis wird wiederum im Akkumulator abgespeichert.

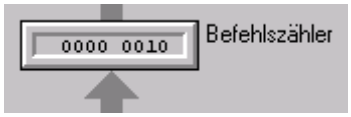
Vorbelegt werden kann der Akkumulator nur durch den Inside CPU-Befehl "Laden". Die Abspeicherung seines Inhaltes erfolgt mit dem Befehl "Speichern". Beim Programmstart befindet er sich also in einem nicht definierten Zustand. Wird ohne vorhergehenden Ladebefehl eine Rechenoperation durchgeführt, ist das Ergebnis nicht vorhersagbar.

Wie bei anderen Registern auch, wird der Akkumulator nicht gelöscht, sondern immer nur überschrieben. Während des Programmlaufes bleibt der Inhalt des Akkumulators solange bestehen, bis er überschrieben wird. Auch nach dem Abspeichern des Inhaltes bleibt er unverändert im Register stehen.

In der Inside CPU hat der Akkumulator eine Größe von 12 Bit. Er kann damit Zahlen im Wertebereich von -2048 bis +2047 aufnehmen (binäre Darstellung: $2^{12} = 4096$).

5.4.3 Befehlszähler (auch Adresszähler genannt)

Eines der wichtigsten Register in jedem Prozessor ist der Befehlszähler oder auch PC-Register genannt (PC steht hier für Program Counter). Er dient dazu, die richtige Reihenfolge der Befehle des Programmablaufes zu sichern.

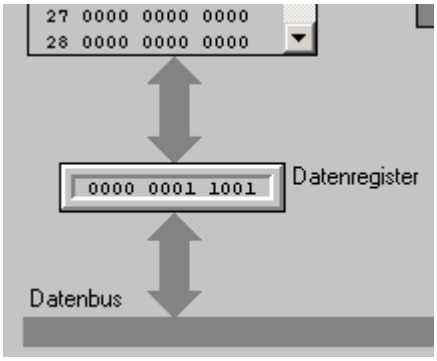


Im Adresszähler steht immer die Speicheradresse, auf die den nächsten Befehl enthält. Nach dem Einschalten der Versorgungsspannung oder nach einem RESET der CPU wird dieses Register immer auf den Wert 0 gesetzt.

5.4.4 Datenregister

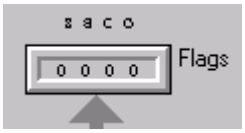
Das Datenregister ist ein spezieller Speicher. Es bildet einen Puffer zwischen dem Datenbus und dem Hauptspeicher. Hier werden aus dem Hauptspeicher angeforderte Daten zunächst zwischengespeichert, um dann über den Datenbus an das entsprechende Register weitergeleitet zu werden.

Andererseits gelangen Daten aus anderen CPU-Registern nur über diesen Zwischenspeicher in den Arbeitsspeicher. Die Größe des Datenregisters entspricht der Größe eines Speicherwortes im Arbeitsspeicher.



5.4.5 Flagregister

Das Flagregister ist ein Register, in dem Zustände abgefragt und gesetzt werden können. Es ist für die Steuerung des Programmablaufs von großer Bedeutung. Der Prozessor speichert bestimmte Ergebnisse arithmetischer Operationen (Addition, Subtraktion, Multiplikation, Division) in den Flags (auch Kennzeichen-Bits oder Merker genannt). Diese Informationen werden für die bedingten Sprungbefehle (Springe wenn 0, Springe wenn negativ) benötigt.



In Inside CPU gibt es folgende Flags:
- Zero-Flag Z (auch Null-Flag genannt)
- Sign-Flag S (auch Vorzeichen-Flag genannt)
- Carry-Flag C (auch Übertragungs-Flag genannt)
- Overflow-Flag O (auch Überlauf-Flag genannt)

Das Zero-Flag (Z)

Das Zero-Flag (Null-Flag) wird immer dann auf 1 gesetzt, wenn das Ergebnis einer Rechnung gleich 0 ist.

Beispiel: 6 - 6 = 0

Dual	Dezimal
0000 0000 0110	6
+ 1111 1111 1010	- 6
0000 0000 0000	0

Sign-Flag (S)

Das Sign-Flag (S) (Vorzeichen-Flag) ist unmittelbar mit dem Bit 12 des Ergebnisses verbunden. Bedenken Sie, dass in der Zweierkomplement-Schreibweise eine 1 in Bit 12 eine negative Zahl bedeutet.

Beispiel: 3 - 5 = -2

Dual	Dezimal
0000 0000 0011	3
- 0000 0000 0101	- 5
1111 1111 1110	- 2

Carryflag (C)

Das Übertragungsbit speichert den arithmetischen Übertrag, d.h. das dreizehnte Bit. Es entsteht als Überlauf einer arithmetischen 12-Bit Operation.

Beispiel: -4 - 1 = -5

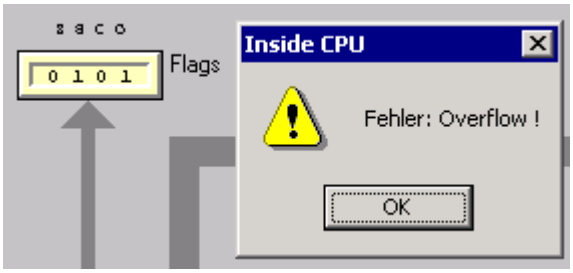
Dual	Dezimal
1111 1111 1100	- 4
- 1111 1111 1111	- 1
1 1111 1111 1011	- 5

Overflow-Flag (O)

Overflow (Überlauf) bedeutet hier, dass ein arithmetischer Übertrag innerhalb eines Wortes den Wert des höchstwertigen Bits verändert hat. Dies führt bei der üblichen Komplementärschreibweise zu einem Vorzeichenfehler. Bit 12 kennzeichnet im Zweierkomplement das Vorzeichen: 1 bedeutet negativ, 0 positiv. Immer wenn zwei Zweierkomplementzahlen addiert werden, kann der Übertrag in das Vorzeichenbit (Bit 12) überlaufen. Geschieht dies, kann sich eine negative in eine positive Zahl wandeln.

Beispiel: 1536 - 768 = -1792

Dual	Dezimal
0110 0000 0000	1536
+ 0011 0000 0000	+ 768
1001 0000 0000	- 1792



5.4.6 Instruktionsregister

Das Instruktionsregister (auch Befehlsregister genannt) hat in PC-Prozessoren in Verbindung mit einer Befehls-Warteschlange (Queue) die Aufgabe, den auszuführenden Befehlscode für die Übergabe an den Decodierer zwischenzuspeichern. Es hat i.d.R. eine Größe von 8 Bit. Je nach Aufbau und Organisation des Arbeitsspeichers ist es möglich, dass der übertragene Speicherinhalt nicht den vollständigen Befehlscode enthält (Mehr-Adressmaschinen). In diesem Fall ist der Befehlscode auf die nachfolgenden Speicherplätze verteilt.

Die hier simulierte CPU ist aus Gründen der Übersichtlichkeit als Ein-Adress-Maschine konzipiert. Der gesamte Befehl lässt sich in einem Speicherplatz unterbringen und kann mit einem Lesezyklus vollständig erfasst werden. Mit Hilfe der im Befehlszähler stehenden Adresse wird die Stelle im Arbeitsspeicher ermittelt, an welcher der nächste auszuführende Befehl steht. Dieser dort stehende Befehlscode wird in das Instruktionsregister übertragen und decodiert. Nach der Decodierung des Befehlscodes wird der Adressteil über den Adressbus zum Adressregister weitergeleitet, von wo aus dann der Zugriff auf den entsprechenden Speicherplatz des Arbeitsspeichers vorgenommen wird. Während des Programmablaufes enthält das Instruktionsregister also immer den Programmbefehl, der gerade ausgeführt wird.

5.4.7 Stackpointer

Der Stackpointer (Stapelzeiger) ist ein Zeiger (Adressspeicherplatz), der die Adresse des obersten Speicherplatzes des Stapelspeichers enthält. Benötigt wird er beim Aufruf eines Unterpro-

gramms zur Speicherung der Rücksprungadresse des aufrufenden Programms.

5.5 Bus

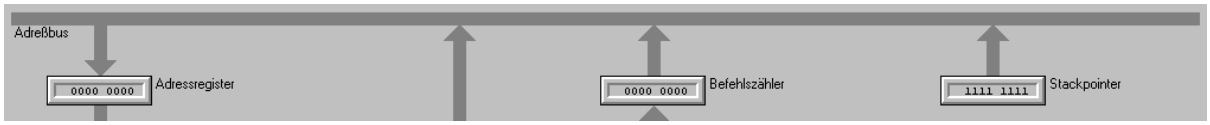
Damit der Mikroprozessor Befehle und Daten vom Arbeitsspeicher verarbeiten kann, müssen diese durch Verbindungsleitungen zum Prozessor selbst übertragen werden. Weiterhin muss eine Verbindung zu den Eingabegeräten (z.B. Tastatur) und Ausgabegeräten (z.B. Bildschirm, Drucker) vorhanden sein, um mit dem Anwender in Kontakt treten zu können. Diese Verbindungswege im Computer werden als Bus-System bzw. Bus bezeichnet.

Bei einem PC werden die folgenden drei Bus-Systeme unterschieden:
- Adressbus
- Datenbus
- Steuerbus

5.5.1 Adressbus

Das Steuerwerk im Mikroprozessor bekommt während der Laufzeit eines Programms Informationen vom Anwendungsprogramm nur über Adressleitungen. Eine Adresse spezifiziert eine Speicherstelle im Arbeitsspeicher. Um an den Inhalt dieser Speicherstelle zu gelangen, wird die Adresse durch das Steuerwerk auf den Adressbus gelegt. Das Adressregister bzw. der Adressbus zeigt dann auf den betreffenden Speicherinhalt im Arbeitsspeicher, der zur Verarbeitung ansteht.

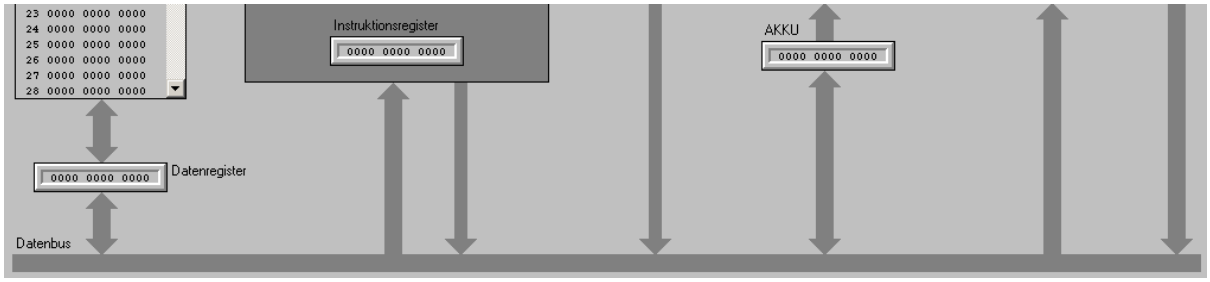
Der Adressbus ist ein unidirektionaler Bus. Das heißt, eine Übertragung von Informationen ist nur von einer Richtung aus möglich (hier: CPU). Durch die Anzahl der einzelnen Leitungen im Adressbus ist die Anzahl der maximal zu adressierenden Speicherstellen begrenzt (Busbreite).



Inside CPU hat eine Adressbreite von 8 Bit, mit der 2^8 = verschiedene Speicherstellen angesprochen werden können.

5.5.2 Datenbus

Unter einem Datenbus versteht man eine Anzahl von Leitungen zur Übertragung von Informationen (Daten) zwischen Rechenwerk (mit dem Akkumulator), Instruktionsregister (inklusive Decodierer und anderen Registern) und Arbeitsspeicher ermöglichen.



Die Schnittstelle zwischen Arbeitsspeicher und Datenbus bildet das Datenregister. Wenn die gewünschte Speicherzelle über den Adressbus ausfindig gemacht worden ist, kann der Inhalt dieser Speicherstelle über den Datenbus verschickt werden. Über den Datenbus werden alle Daten aus dem Arbeitsspeicher transportiert. Diese Daten werden anschließend im Prozessor weiterverarbeitet und bei Bedarf wieder zum Arbeitsspeicher zurücktransportiert.

Ob es sich bei diesen Informationen um Programmbefehle handelt oder um Operanden, die im Datenteil des Programms definiert wurden (Daten), ist bei Mikroprozessoren ohne Bedeutung. Der Einfachheit halber werden Code und Daten über den gleichen Datenbus transportiert.

Über den Datenbus kann der Prozessor somit Informationen lesen und nach der Verarbeitung wieder zurückschreiben. Die Datenübertragung verläuft hier bidirektional.

5.5.3 Steuerbus

Der Steuerbus übernimmt bei der Datenübertragung im Computer die Rolle eines Verkehrspolizisten auf einer stark befahrenen Kreuzung. In welcher Richtung die Informationen transportiert werden, wird über den Steuerbus geregelt. Natürlich wird der Steuerbus hierbei von anderen Bauteilen mit entsprechenden Informationen versorgt.

Eine Anweisung, eine Speicherstelle zu lesen oder zu beschreiben, wird vom Steuerwerk des Prozessors über den Steuerbus weitergegeben. Ist der Arbeitsspeicher nun beispielsweise zur Ein- und Ausgabe von Speicherinhalten nicht bereit, wendet sich dieser an den Steuerbus. Der Steuerbus sperrt dann den Zugriff auf den Arbeitsspeicher. Ist der Arbeitsspeicher wieder frei, wird die Blockierung vom Steuerbus aufgehoben, und es können weiter Daten ausgetauscht werden. Der Steuerbus koordiniert also die ganzen zeitlichen Abläufe, damit es nicht zu einem Stau oder noch schlimmer zu einem Datencrash kommt.

Um die Unübersichtlichkeit nicht zu beeinträchtigen, wird der Steuerbus in Inside CPU nicht dargestellt.

5.5.4 Takt

Das Steuerwerk als wichtigster Teil des Prozessors überhaupt steuert mit Hilfe von Taktsignalen die Folge der Ereignisse, die zur Ausführung der einzelnen Befehle notwendig ist. In Abhängigkeit vom Befehlscode werden über eine Vielzahl von Leitungen Impulse an Geräte und Register ausgegeben, Daten- und Adresswege geöffnet und geschlossen, Lese- und Schreibvorgänge ausgelöst und vieles mehr. Die Abläufe in der CPU sind zyklisch, d.h. sie wiederholen sich Befehl für Befehl. Die Anzahl der Befehlstakte (Schritte) variiert, so dass längere und kürzere Befehlsfolgen entstehen.

Im Simulationsfenster von Inside CPU lassen sich diese Befehlsschritte verfolgen. Der Taktfrequenz kann durch die Schaltflächen „sehr langsam“, „langsam“, „mittel“, „schnell“, „sehr schnell“ verändert werden. Wiederum aus Gründen der Übersichtlichkeit wurde auf sämtliche Takt- und Steuerleitungen in der grafischen Darstellung verzichtet.

6 Programmerstellung in Inside CPU

Inside CPU ermöglicht es Ihnen, eigene Programme selbst zu erstellen und ihren Ablauf zu beobachten. Die Anweisungen (Befehle) dieser Programme sind naturgemäß maschinenorientiert und ähneln Assembler-Befehlen. Der komfortable Editor erleichtert die visuelle Programmierung ganz erheblich. Folgende Hinweise sollte man jedoch beachten:

- Das Programm sollte grundsätzlich in der Adresse "00" des Arbeitsspeichers beginnen, da der Befehlszähler mit 0 vorbelegt ist.
- Der erste Programmbefehl ist in der Regel "Laden", um Daten in den Akkumulator zu bringen.
- Der Anweisungsteil eines Programms (in Assembler: Code-Segment), in dem alle Befehle abgelegt sind, sollte im Arbeitsspeicher einen zusammenhängenden Bereich einnehmen.
- Der letzte Programmbefehl (im Hauptprogramm) muss der "Stop"-Befehl sein, im Unterprogramm "Springe zurück".
- Die untersten Adressen des Arbeitsspeichers sind frei zu halten, da sie als Stack für die Unterprogramm-Rücksprungadressen benutzt werden.
- Der Datenbereich enthält alle für das Programm erforderlichen Eingabedaten. Für Speicherung von Zwischenergebnissen und Ausgabedaten ist hier ebenfalls Speicherplatz vorzusehen.
- Zur besseren Übersicht (und zur Fehlervermeidung) sind Programmbereich und Datenbereich voneinander zu trennen.

Ein Programm wird als Folge binärer Befehle im Arbeitsspeicher gespeichert, die in aufeinanderfolgenden Adressen liegen. Programmbefehle werde sequentiell im Speicher abgelegt, jedoch nicht unbedingt in dieser Reihenfolge abgearbeitet.

Um je nach Ergebnis einer Operation unterschiedliche Aktionen durchzuführen, können andere Programmteile bearbeitet werden. In diesem Fall bestimmt der gerade ausgeführte Befehl den als nächsten auszuführenden. Ein Befehl, der den sequentiellen Programmablauf unterbricht, bewirkt eine Verzweigung (z.B. Springe wenn 0). So kann der Programmierer einen Sprung zu einer bestimmten Speicherstelle angeben („Wenn die letzte Berechnung 0 ergeben hat, dann springe zur Adresse x, sonst tue nichts“). Ein derartiger Befehl erzwingt einen neuen Wert im Programmzähler.

6.1 Inside CPU-Befehle

Inside CPU beinhaltet folgende Befehle:

6.1.1 Ladebefehl (Lade)

Der Befehl "Laden" bewirkt, dass der Arbeitsspeicher unter der angegebenen Adresse gelesen und der darin stehende Speicherinhalt in den Akkumulator gebracht wird. Der Speicherinhalt selbst bleibt hiervon unberührt und besteht unverändert weiter.

Code	Adresse	Befehl	Dezimal
0001	0000 1000	Lade	8

Ist dies nicht der erste Programmbefehl, so ist darauf zu achten, ob im Akkumulator ein Wert steht, z.b. ein Ergebnis aus einer vorherigen Rechenoperation, der noch nicht gesichert ist. Durch den Ladebefehl wird nämlich der alte Inhalt des Akkumulators überschrieben und steht nicht mehr zur Verfügung. In einem solchen Fall müsste der alte Akkuinhalt vor dem Ladebefehl mit dem Befehl "Speichern" gesichert werden.

Benötigt wird der Ladebefehl für alle hier durchführbaren Rechenoperationen, um die erste Zahl (erster Operand) in das Rechenwerk (Akku) zu transportieren. Der zweite Zahl, die an der Rechnung beteiligt ist (das ist der 2. Operand), wird dem Rechenwerk automatisch durch den Rechenbefehl (z.B. Addieren, Subtrahieren, Dividieren, Multiplizieren) zugeführt.

Den Ladebefehl kann man auch dazu benutzen, um den Inhalt eines Speicherplatzes in einen anderen Speicherplatz zu schreiben.

6.1.2 Speichern-Befehl (Speichere)

Der Befehl „Speichern“ ist das Gegenstück zum „Laden“. Er bringt den jeweiligen Inhalt des Akkumulators an die Stelle im Arbeitsspeicher, die durch die Adresse im Speicherbefehl angegeben wurde.

Code	Adresse	Befehl	Dezimal
0010	0000 1010	Speichere	10

Er dient somit zur Sicherung von Rechenergebnissen, die nach ausgeführter Rechenoperation im Akkumulator stehen. Der Inhalt des Akkumulators bleibt auch nach dem Speicherbefehl unverändert dort stehen, bis er durch einen Lade- oder Rechenbefehl (Addiere, Subtrahiere, Multipliziere, Dividiere) überschrieben wird.

6.1.3 Additionsbefehl (Addiere)

Der Befehl „Addieren“ lädt den Speicherinhalt unter der im Adressteil angegebenen Adresse in den Speicherplatz des Rechenwerks (Operandenregister), addiert ihn dann zum Inhalt des Akkumulators und speichert das Ergebnis wieder im Akkumulator.

Code	Adresse	Befehl	Dezimal
0011	0000 1001	Addiere	9

Beide an der Rechnung beteiligten Zahlen (Operanden) stehen nach der Ausführung des Additionsbefehls dem Rechenwerk nicht mehr zur Verfügung. Der erste Operand wird im Akkumulator vom Ergebnis überschrieben. Der zweite Operand im Operandenregister ist ohnehin nicht

direkt ansprechbar und wird beim nächsten Rechenbefehl überschrieben.

6.1.4 Subtraktionsbefehl (Subtrahiere)

Der Befehl „Subtrahieren“ lädt den Speicherinhalt unter der im Adressteil angegebenen Adresse in den Speicherplatz des Rechenwerks (Operandenregister), subtrahiert ihn dann zum Inhalt des Akkumulators und speichert das Ergebnis wieder im Akkumulator.

Code	Adresse	Befehl	Dezimal
0100	0001 0101	Subtrah.	21

Beide an der Rechnung beteiligten Zahlen (Operanden) stehen nach der Ausführung des Subtraktionsbefehls dem Rechenwerk nicht mehr zur Verfügung. Im Gegensatz zur Addition ist darauf zu achten, welchen Operanden man in den Akkumulator und welchen man durch den Subtraktionsbefehl in das Operandenregister des Rechenwerkes lädt, weil die Subtraktion nicht kommutativ ist.

6.1.5 Multiplikationsbefehl (Multipliziere)

Der Befehl "Multiplizieren" lädt den Speicherinhalt unter der im Adressteil angegebenen Adresse in den Speicherplatz des Rechenwerks (Operandenregister), multipliziert ihn dann mit dem Inhalt des Akkumulators und legt das Ergebnis wieder im Akkumulator ab.

Code	Adresse	Befehl	Dezimal
0101	0000 1101	Multipl.	13

Beide an der Rechnung beteiligten Zahlen (Operanden) stehen - wie bei der Addition - nach Ausführung des Multiplikationsbefehls dem Rechenwerk nicht mehr zur Verfügung.

6.1.6 Divisionsbefehl (Dividiere)

Der Befehl "Dividieren" lädt den Speicherinhalt unter der im Adressteil angegebenen Adresse in den Speicherplatz des Rechenwerks (Operandenregister), dividiert den Inhalt des Akkumulators durch diesen Speicherinhalt und legt den ganzzahligen Teil des Ergebnisses im Akkumulator ab. Divisionsreste können nicht gespeichert werden.

Code	Adresse	Befehl	Dezimal
0110	0001 0001	Dividiere	17

Beide an der Rechnung beteiligten Zahlen (Operanden) stehen - wie bei der Addition - nach Ausführung des Divisionsbefehls dem Rechenwerk nicht mehr zur Verfügung. Im Gegensatz zur Multiplikation ist hier darauf zu achten, welchen Operanden man in den Akkumulator und welchen man durch den Divisionsbefehl in das Operandenregister des Rechenwerks lädt, weil die Division nicht kommutativ ist.

Eine Division durch Null wird vor ihrer Ausführung erkannt und durch eine entsprechende Fehlermeldung angezeigt.

6.1.7 Sprung-Befehl

Bei einer linearen Programmabarbeitung kommt nach einem Befehl im aktuellen Speicherplatz n stets der Befehl im nächst höheren Speicherplatz n+1. Sprungbefehle realisieren einen nicht-linearen Programmablauf. Springen bzw. Verzweigen bedeutet, dass der nächste Befehl eben nicht vom Nachfolger (n+1) der aktuellen Adresse (n) geholt wird. Das automatische Inkrementieren (erhöhen um 1) des Befehlszählers (Programmzählers) in der Decodierphase reicht dann nicht mehr aus. Sprungbefehle sind für die Programmierung sehr wichtig, da sie Verzweigungen, Schleifen und Unterprogramme ermöglichen.

Inside CPU stellt fünf Sprungbefehle zur Verfügung.

Springe

Dieser unbedingte Sprungbefehl wird immer ausgeführt.

Code	Adresse	Befehl	Dezimal
0111	1101 1010	Springe	218

Der Prozessor führt den Befehl als nächsten aus, den er unter der im Sprungbefehl angegebenen Adresse findet. Dazu wird der Befehlszähler mit dieser Adresse überschrieben. Nach dem Sprungbefehl erfolgt keine Rückkehr zur ursprünglichen Position, wie es bei „Springe in Unterprogramm“ (Verzweigen zum Unterprogramm) geschieht.

Springe, wenn 0

Dieser bedingte Sprungbefehl wird ausgeführt, wenn das Ergebnis im AKKU gleich 0 und das Zero-Flag (Z) auf „1“ gesetzt worden ist.

Code	Adresse	Befehl	Dezimal
1000	1101 1010	Springe 0	218

Der bedingte Befehl "Springen wenn 0" wird nur unter der Voraussetzung ausgeführt, dass der aktuelle Wert, der im Akkumulator steht, 0 ist und im Flagregister das Zero-Flag (Z) auf „1“ gesetzt worden ist. Ist er positiv oder negativ, wird der Sprungbefehl ignoriert und mit dem Befehl fortgefahren, der in der nächsten Speicherstelle steht.

Springe, wenn negativ

Dieser bedingte Sprungbefehl wird ausgeführt, wenn das Ergebnis im AKKU negativ und das Sign-Flag (S) auf „1“ gesetzt worden ist.

Code	Adresse	Befehl	Dezimal
1001	1101 1010	Springe -	218

Der bedingte Befehl "Springen wenn negativ" wird nur unter der Voraussetzung ausgeführt, dass der aktuelle Wert, der im Akkumulator steht, negativ ist und im Flagregister das Sign-Flag (S) auf „1“ gesetzt worden ist. Ist er positiv oder 0, wird der Sprungbefehl ignoriert und mit dem Befehl fortgefahren, der in der nächsten Speicherstelle steht.

Springe in Unterprogramm

Dieser unbedingte Sprungbefehl wird immer ausgeführt.

Code	Adresse	Befehl	Dezimal
1010	1101 1010	Spr. Upro	218

Der Befehl "Springen ins Unterprogramm" veranlasst einen Sprung zu der im Befehl angegebenen Adresse. Zuvor wird jedoch die Arbeitsspeicheradresse, die im Befehlszähler steht, in der letzten, noch nicht belegten Speicherstelle des Arbeitsspeichers (Stapelspeicher) gesichert, um nach Abarbeitung des Unterprogramms mit dem Rücksprungbefehl zu der Ausgangsposition im Hauptprogramm zurückkehren zu können.

Im Unterprogramm selbst können beliebig viele Befehle codiert werden, sogar weitere Unterprogrammaufrufe. Damit der Prozessor das Ende des Unterprogramms erkennen kann, muss das Unterprogramm immer mit einem Rücksprungbefehl abgeschlossen werden.

Springe zurück

Dieser unbedingte Sprungbefehl wird immer ausgeführt.

Code		Befehl	Dezimal
1011	0000 0000	Rückspr.	-

Mit dem Befehl "Rücksprung" springt man aus dem Unterprogramm an die Adresse des Hauptprogramms zurück, an der in das Unterprogramm verzweigt worden. Er beendet grundsätzlich ein Unterprogramm und muss in jedem Falle dort (und nur dort!) codiert sein, andernfalls erscheint eine Fehlermeldung. Der Rücksprungbefehl zeigt dem Prozessor an, dass ein Unterprogramm, das durch den Befehl "Springen in Unterprogramm" aufgerufen wurde, jetzt abgearbeitet ist und die Rückkehr ins Hauptprogramm erfolgen soll.

Die dazu erforderliche Rücksprungadresse wurde beim Aufruf des Unterprogramms auf einem speziellen Speicherplatz im Arbeitsspeicher (Stapelspeicher) hinterlegt, auf den der Stackpointer zeigt. Über diesen Stackpointer kann sie jetzt wieder in den Befehlszähler zurückgebracht werden. Damit kann das aufrufende Programm hinter dem Unterprogrammaufruf fortgesetzt werden. Da die Rücksprungadresse aus dem Arbeitsspeicher geholt wird, ist bei diesem Befehl ein Adressteil nicht erforderlich.

6.1.8 Stopbefehl (Stop)

Der Befehl "Stop" muss am Ende eines jeden Programms stehen, damit der Prozessor keine weiteren Versuche unternimmt, Befehle zu lesen. Der Arbeitsspeicher ist aus diesem Grunde zu Beginn mit Stopbefehlen vorbelegt. Er benötigt keinen Adressteil.

Code		Befehl	Dezimal
0000	0000 0000	Stop	-

Der Stopbefehl darf niemals ein Unterprogramm beenden, da der Prozessor sonst den Programmablauf abbrechen würde und nicht mehr in das Hauptprogramm zurückkehren könnte.

6.2 Schleifen

Wiederholungen können - wie in Assembler - durch Sprungbefehle realisiert werden. Die von den höheren Programmiersprachen her bekannten Schleifenarten (kopfgesteuerte WHILE-Zählschleifen, fußgesteuerte REPEAT-Schleifen, Zählschleifen mit FOR...NEXT) sind dabei auf eine sehr einfache Struktur mit der bedingten Sprunganweisung "Springe wenn negativ" zu reduzieren.

Einfache Zählschleifen, die einen Laufindex von 3 herunterzählen, findet Sie in den Beispielprogrammen Dekrementieren1.sim und Dekrementieren2.sim.

6.3 Unterprogramme

Wenn in einem Programm eine bestimmte Befehlsfolge an verschiedenen Stellen auftritt, so bietet es sich an, diese Befehlsfolge ein einziges Mal in einem Unterprogramm zu codieren. Mit Hilfe des Befehls "Springe in Unterprogramm" wird diese Befehlsfolge aufgerufen, wenn sie gebraucht wird.

7 Anhänge

7.1 Datei WIN_CPU.INI anpassen

Nach der Installation von CPU Simulation befindet sich in Ihrem Windows Verzeichnis die Datei: *WIN_CPU.INI* . Sie enthält einige Voreinstellungen für CPU Simulation, welche Sie nach Ihren Bedürfnissen modifizieren können.

In der WIN_CPU.INI finden Sie folgende Sektionen:

[General]
[Sound]
[Printer]
[Editor]
[Simulation]
[Colors]

Sektion [General]

In der Sektion [General] sind keine benutzerdefinierten Einstellungen durchzuführen!

[General]
IDString=4621
Kennung (nicht ändern!).

Sektion [Sound]

In der Sektion [Sound] finden Sie Einstellungen für Klangunterstützung. Die Klangunterstützung funktioniert nur, wenn ein Klangtreiber in Windows installiert ist.

[Sound]
SoundSupport=TRUE
Wenn dieser Schalter auf TRUE gesetzt ist wird es dem Benutzer ermöglicht die Klangunterstützung einzuschalten.

SoundEnabled=TRUE
Wenn dieser Schalter auf TRUE gesetzt ist dann ist die Klangunterstützung beim Laden von CPU Simulation eingeschaltet. Die Klangunterstützung lässt sich im Simulationsfenster von CPU Simulation ebenfalls ein / aus - schalten.

TickSound=.\TICK.WAV
Klangdatei, welche beim Klicken von Schaltern im Simulationsfenster ertönt.

RegOutSound=.\reg_o.wav
Klangdatei, welche während der Simulation beim Lesen von Registern ertönt.

RegInSound=.\reg_i.wav
Klangdatei, welche während der Simulation beim Schreiben von Registern ertönt.

ErrorSound=.\error.wav
Klangdatei, welche während der Simulation beim Auftreten von Programmfehlern ertönt.

Sektion [Printer]

In der Sektion [Printer] finden Sie Einstellungen für das Drucken von Anwendungsprogrammen.

[Printer]
FontName=Courier New
Angabe welcher Font beim Drucken von Anwendungsprogrammen benutzt werden soll. Es empfiehlt sich hier nur nicht proportionale Fonts anzugeben (z.B. *Courier, Courier New, Terminal, Fixedsys*), da sonst die Spaltenformatierung verloren geht.

FontSize=10
Gibt die Fontgröße an mit der gedruckt werden soll.

Sektion [Editor]

In der Sektion [Editor] finden Sie Einstellungen für das Erscheinungsbild des Editors.

[Editor]
Maximized=TRUE
Bewirkt, dass das Editorfenster bei Programmstart Maximiert wird.

Left=1188
Gibt den Abstand des Editorfensters zum linken Rand des Bildschirms in Twips an.

Top=1142
Gibt den Abstand des Editorfensters zum oberen Rand des Bildschirms in Twips an.

Load=.\DEMO.SIM
Hier können Sie eine CPU Simulation Datei angeben, welche bei jedem Programmstart in den Editor geladen wird. Falls die angegebene Datei bei Programmstart nicht existiert wird sie automatisch erzeugt.

UserProgramDirectory=.\PROGRAM
Gibt das Verzeichnis an, in dem Anwendungsprogramme abgelegt werden.

Sektion [Simulation]

In der Sektion [Simulation] finden Sie Einstellungen für Eigenschaften und Erscheinungsbild des Simulationsfensters.

[Simulation]
Maximized=TRUE
Bewirkt, dass das Simulationsfenster bei Aufruf Maximiert wird.

Left=1418
Gibt den Abstand des Simulationsfensters zum linken Rand des Bildschirms in Twips an.

Top=509
Gibt den Abstand des Simulationsfensters zum oberen Rand des Bildschirms in Twips an.

DefaultSpeed=1
Gibt an, welche Simulationsgeschwindigkeit bei Programmstart voreingestellt ist.
(0 ... 4, 0 = sehr langsam)

KeepBusHighlighted=TRUE
Bewirkt, dass ein soeben abgearbeiteter Bus bis zum Beginn des nächsten Befehlszyklus erhellt bleibt.

Bus12XSize=8
Breite der vertikalen 12 Bit Busse (Datenbus).

Bus12YSize=8
Breite der horizontalen 12 Bit Busse (Datenbus).

Bus8XSize=6
Breite der vertikalen 8 Bit Busse (Adressbus).

Bus8YSize=6
Breite der horizontalen 8 Bit Busse (Adressbus).

Bus4XSize=4
Breite der vertikalen 4 Bit Busse (Bus zu den Flags).

Bus4YSize=4
Breite der horizontalen 4 Bit Busse.

ArrowWidth=12
Breite der Pfeilspitzen.

ArrowLength=12
Länge der Pfeilspitzen.

Sektion [Colors]

In der Sektion [Colors] finden Sie Einstellungen für die Farben der CPU Darstellung im Simulationsfenster.

[Colors]
NormalRegisterColor=12632256
Farbe der Register.

SourceRegisterColor=8519679
Farbe der Register, aus denen gelesen wird.

DestinationRegisterColor=12582911
Farbe der Register, in die geschrieben wird.

FormerRegisterColor=121048
Farbe der Register, in die beim letzten Zyklus geschrieben wurde.

LoBusColor= 8421504
Farbe der Busse.

HiBusColor=8454143
Farbe der animierten Busse.

LoArrowColor=8421504
Farbe der Pfeilspitzen.

HiArrowColor=8454143
Farbe der animierten Pfeilspitzen.

ALUColor=8421504
Farbe der Arithmetic Logical Unit.

DecoderColor=8421504
Farbe des Decoders.

RamColor=12632256
Farbe des Hauptspeichers.

7.2 ASCII-Tabelle

DEZ	HEX	Zeichen	DEZ	HEX	Zeichen	DEZ	HEX	Zeichen	DEZ	HEX	Zeichen
00	00	NUL	32	20	SP	64	40	@	96	60	`
01	01	SOH	33	21	!	65	41	A	97	61	a
02	02	STX	34	22	"	66	42	B	98	62	b
03	03	ETX	35	23	#	67	43	C	99	63	c
04	04	EOT	36	24	$	68	44	D	100	64	d
05	05	ENQ	37	25	%	69	45	E	101	65	e
06	06	ACK	38	26	&	70	46	F	102	66	f
07	07	BEL	39	27	'	71	47	G	103	67	g
08	08	BS	40	28	(	72	48	H	104	68	h
09	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

DEZ	HEX	Zeichen	DEZ	HEX	Zeichen	DEZ	HEX	Zeichen	DEZ	HEX	Zeichen
128	80	(CTRL)	160	A0	NB SP	192	C0	À	224	E0	à
129	81	(CTRL)	161	A1	¡	193	C1	Á	225	E1	á
130	82	(CTRL)	162	A2	¢	194	C2	Â	226	E2	â
131	83	(CTRL)	163	A3	£	195	C3	Ã	227	E3	ã
132	84	(CTRL)	164	A4	¤	196	C4	Ä	228	E4	ä
133	85	(CTRL)	165	A5	¥	197	C5	Å	229	E5	å
134	86	(CTRL)	166	A6	¦	198	C6	Æ	230	E6	æ
135	87	(CTRL)	167	A7	§	199	C7	Ç	231	E7	ç
136	88	(CTRL)	168	A8	¨	200	C8	È	232	E8	è
137	89	(CTRL)	169	A9	©	201	C9	É	233	E9	é
138	8A	(CTRL)	170	AA	ª	202	CA	Ê	234	EA	ê
139	8B	(CTRL)	171	AB	«	203	CB	Ë	235	EB	ë
140	8C	(CTRL)	172	AC	¬	204	CC	Ì	236	EC	ì
141	8D	(CTRL)	173	AD		205	CD	Í	237	ED	í
142	8E	(CTRL)	174	AE	®	206	CE	Î	238	EE	î
143	8F	(CTRL)	175	AF	¯	207	CF	Ï	239	EF	ï
144	90	(CTRL)	176	B0	°	208	D0	Đ	240	F0	ð
145	91	(CTRL)	177	B1	±	209	D1	Ñ	241	F1	ñ
146	92	(CTRL)	178	B"	²	210	D2	Ò	242	F2	ò
147	93	(CTRL)	179	B3	³	211	D3	Ó	243	F3	ó
148	94	(CTRL)	180	B4	´	212	D4	Ô	244	F4	ô
149	95	(CTRL)	181	B5	µ	213	D5	Õ	245	F5	õ
150	96	(CTRL)	182	B6	¶	214	D6	Ö	246	F6	ö
151	97	(CTRL)	183	B7	·	215	D7	×	247	F7	÷
152	98	(CTRL)	184	B8	¸	216	D8	Ø	248	F8	ø
153	99	(CTRL)	185	B9	¹	217	D9	Ù	249	F9	ù
154	9A	(CTRL)	186	BA	º	218	DA	Ú	250	FA	ú
155	9B	(CTRL)	187	BB	»	219	DB	Û	251	FB	û
156	9C	(CTRL)	188	BC	¼	220	DC	Ü	252	FC	ü
157	9D	(CTRL)	189	BD	½	221	DD	Ý	253	FD	ý
158	9E	(CTRL)	190	BE	¾	222	DE	Þ	254	FE	þ
159	9F	(CTRL)	191	BF	¿	223	DF	ß	255	FF	ÿ