“Funktionsweise eines Rechners” in der 12. Jahrgangsstufe mit dem Mikrocontroller

Christoph Krichenbauer

schule@krichenbauer.de

14. März 2014

Inhaltsverzeichnis

1 Idee

Eine Simulation einer Registermaschine ist wohl üblich in der 12. Jahrgangstufe. Doch warum sollte man Dinge simulieren, die auch im Original von den Schülern verstanden werden können? Es erscheint mit etwas Absurd einen einfachen Modellcomputer auf einen komplizierten Computer zu simulieren, nur um damit zu erklären, dass der komplizierte Computer im Prinzip genauso funktioniert wie der simulierte einfache. Doch welche Architektur ist heutzutage noch einfach genug, um in einer Unterrichtssequenz besprochen zu werden?

Während zu Zeiten des C-64 noch durch wenige Zeilen Assemblercode bereits sichtbare Ergebnisse auf dem Bildschirm zu bringen waren, sind heutzutage sehr individuelle und komplexe Aufrufe nötig, um den Grafikcontroller anzusprechen. Somit scheiden handelsübliche PCs für den Einsatz in diesem Feld aus.

Naheliegend wäre die Nutzung historische Systeme, wie die besagte 6502-Architektur des C-64 oder Apple II, jedoch ist auch hier ein breites Vorwissen über die Ansteuerung des Videocontrollers nötig. In der kurzen Zeit der Unterrichtssequenz ließen sich so nur bescheidene Ergebnisse erreichen, deren Motivation fraglich ist. Auch ist der nötige Aufwand, auch nur einen halben Klassensatz C-64-Arbeitsplätze zu betreiben, unangemessen. Besagte Systeme nur zu simulieren, schafft auch keine Vorteile gegenüber den Registermaschinensimulatoren.

Somit sind noch einfachere Systeme für den Unterrichtseinsatz notwendig: Mikrocontroller. Diese sind von verschiedenen Herstellern in zahlreichen Varianten verfügbar und werden sowohl in professionellen Anwendungen als auch von Bastlern im privaten Umfeld in großen Stückzahlen genutzt. Dabei reicht der Anwendungsbereich von der automatisierten Steuerung der Modellbahnanlage über elektronische Türschlosssysteme bishin zu Steuersystemen von Kaffeemaschinen. Praktisch in allen elektronischen Geräten werden aufgrund der geringen Kosten Mikrocontroller verbaut.

Die intensive Besprechung von Mikrocontrollern im Unterricht soll bei den Schülern somit nicht nur einen praxisorientierten Lehrgang zur Funktionsweise von Rechnern bieten, sondern zudem dazu motivieren, sich selbst mit Mikrocontrollersteuerungen zu beschäftigen. Damit ist eine der Anforderungen an den verwendeten Chip die breite Verfügbarkeit zusätzlicher, an Anfänger gerichtete Dokumentation im Internet.

Somit bietet dieser Ansatz auch einen Einblick in den Fachbereich der Elektrotechnik, der ansonsten weder im Rahmen des Physik-, noch des Informatikunterrichts ausreichend berücksichtigt wird.

2 Funktionsweise eines Rechners am Beispiel von Mikrocontrollern

2.1 Anforderungen des Lehrplans

Nachfolgend der Auszug aus dem Lehrplan im Wortlaut:

> **Inf 12.3 Funktionsweise eines Rechners (ca. 17 Std.)**
>
> Am Modell der Registermaschine lernen die Schüler den grundsätzlichen Aufbau eines Computersystems und die Analogie zwischen den bisher von ihnen verwendeten Ablaufmodellen und Maschinenprogrammen kennen. So wird ihnen auch bewusst, dass Möglichkeiten und Grenzen theoretischer algorithmischer Berechnungsverfahren für die reale maschinelle Verarbeitung von Information ebenfalls gelten.
>
> Beispiele zeigen den Schülern, wie einfache Algorithmen auf systemnaher Ebene durch Maschinenbefehle realisiert werden können. Dabei beschränken sich Anzahl und Komplexität der benutzten Maschinenbefehle auf das für die Umsetzung der gewählten Beispiele Wesentliche. Für das Verstehen des Programmablaufs ist insbesondere die in Jahrgangsstufe 10 erlernte Zustandsmodellierung eine große Hilfe. Zur Überprüfung ihrer Überlegungen setzen die Schüler eine Simulationssoftware für die Zentraleinheit ein, die die Vorgänge beim Programmablauf veranschaulicht.
>
> - Aufbau eines Computersystems: Prozessor (Rechenwerk, Steuerwerk), Arbeitsspeicher, Ein- und Ausgabeeinheiten, Hintergrundspeicher; Datenbus, Adressbus und Steuerbus
> - Registermaschine als Modell eines Daten verarbeitenden Systems (Datenregister, Befehlsregister, Befehlszähler, Statusregister); Arbeitsspeicher für Programme und Daten (von-Neumann-Architektur), Adressierung der Speicherzellen
> - ausgewählte Transport-, Rechen- und Steuerbefehle einer modellhaften Registermaschine; grundsätzlicher Befehlszyklus
> - Zustandsübergänge der Registermaschine als Wirkung von Befehlen

- Umsetzung von Wiederholungen und bedingten Anweisungen auf Maschinenebene

Der Lehrplan gibt explizit vor, die Funktionsweise von Rechnern nicht an realen Maschinen zu erarbeiten. Angesichts der hohen Komplexität der heute üblichen Systemarchitekturen ist diese Vorgehensweise nachvollziehbar. Insbesondere in Hinblick auf mögliche Aufgabenstellungen im schriftlichen Abitur ist eine Reduzierung der Komplexität sowie eine Vermeidung der Streuung auf verschiedene Systeme in unterschiedlichen Lehrgängen notwendig. Die Arbeit mit den Schülern an einem realen System kann also nur als Erweiterung erfolgen, ohne die geforderte Modellregistermaschine zu vernachlässigen. Diese zusätzliche Transferleistung kann in einem so kurzen Zeitraum nur möglich sein, wenn Aufbau und Befehlssatz der realen CPU den Registermaschinen bereits möglichst ähnlich sind. Ähnliche Anforderungen sind an die Software zur Programmierung zu stellen. Es muss den Schülern auch bei realen Programmen möglich sein, die Wirkung von Befehlen auf die Zustände von Speicher und CPU nachzuvollziehen. Eine Einzelschrittsteuerung ist also ebenso notwendig wie zumindest ein Lesezugriff auf Register und Arbeitsspeicher.

2.2 Vorüberlegungen

Eine Beschäftigung mit Mikrocontrollern zur Besprechung der vorgegeben Lernziele wirkt auf den ersten Blick wie eine den Schülern nicht zumutbare Zusatzbelastung. Dabei darf jedoch nicht vergessen werden, dass die Arbeit an realer Hardware mit sichtbaren Ergebnissen ungleich motivierender wirkt als die reine Simulation von abstrakten, beispielhaften Rechenaufgaben.
Mit der verwendeten Hard- und Software (siehe 2.3) steht zudem eine Umgebung zur Verfügung, die zwar aufgrund ihrer professionellen Auslegung ungleich komplexer zu steuern ist als die zur Verfügung stehenden Simulatoren, jedoch mit nur kleinen Einschränkungen alle an Simulatoren gestellten Aufgaben erfüllen kann.

Generell sind zwei unterschiedliche Herangehensweisen denkbar. Nachdem die modellhafte Registermaschine zwingend behandelt werden muss, wäre eine Möglichkeit, diese vorzuziehen und auch einfache Aufgaben daran nachzuvollziehen, um danach auf den Mikrocontroller umzusteigen, um komplexere Aufgabenstellungen erfüllen zu können. Dieses Vorgehen zwingt jedoch die Schüler, sich während der Unterrichtssequenz auf neue Gegebenheiten und Software einzulassen, die deutlich komplexer als die

Registermaschine sowie deren Simulatoren ist.
Die Alternative dazu ist, direkt mit dem Mikrocontroller als betrachtete Hardware einzusteigen und die Transferleistung erst gegen Ende der Sequenz auf die einfachere Registermaschine zu legen. Dabei ist von Vorteil, dass bereits zu Beginn des Projektes erste real sichtbare Ergebnisse vorliegen, welche die Schüler dazu motivieren, auch mit schwierigeren Inhalten fortzufahren. Die Abschlussmotivation, sich nach Erarbeitung und Übung nochmal in eine andere, rein theoretische Maschinenarchitektur einzudenken, sollte von den Schülern mit der Aussicht auf mögliche Abituraufgaben aufgebracht werden können. Die hier beschriebene Unterrichtssequenz folgt diesen Überlegungen.

Um interessierten Schülern einen einfachen Einstieg in die Auseinandersetzung mit der Progammierung von Mikrocontrollern über den Unterricht hinaus zu ermöglichen, sind zusätzliche Anforderungen an die Hardware nötig. Einerseits sollten Controller und Programmierplattform erschwinglich sein, andererseits sollten die vom Controller steuerbaren Ein- und Ausgabegeräte so einfach wie möglich sein, da allzu komplizierte Schaltungen eine abschreckende Wirkung haben können.

2.3 Verwendete Hard- und Software

Es gibt eine ganze Reihe an Mikrocontrollern, die aufgrund ihrer Hardwarearchitektur für den Unterricht in Frage kommen. Die meisten verfügbaren Controller sind nach der von-Neumann-Architektur aufgebaut.
Die Wahl fiel jedoch aus einigen Gründen auf den Texas Instruments MSP430. Einerseits ist er, zusammen mit einem USB-Programmer, dem "MSP430 Launchpad[1]", sehr kostengünstig zu erstehen. Ein viel wichtigeres Kriterium ist jedoch die von TI zur Verfügung gestellte Entwicklungsumgebung, das Code Composer Studio[2]. Dabei handelt es sich um eine proprietäre Erweiterung für die bekannte Entwicklungsumgebung Eclipse. Sie bietet einen einfachen Zugriff auf die auf dem Launchpad vorhandene JTAG[3] bzw. Spy-Bi-Wire[4]-Umgebung und bietet somit die Möglichkeit, die auf dem Controller in Ausführung befindlichen Programme zu pausieren, in Einzelschritten durchzuführen sowie den aktuellen Zustand aller Register

[1]Webseite des Texas Instruments Launchpad: http://www.ti.com/launchpad

[2]Webseite des Texas Instruments Code Composer Studios: http://www.ti.com/tool/ccstudio

[3]Joint Test Action Group, Verfahren um in bereits in Arbeitsumgebungen eingebaute Controller auszulesen bzw. zu debuggen.

[4]Spy-Bi-Wire ist das von Texas Instruments entwickelte serielle Protokoll für die JTAG-Schnittstelle

und des Speichers einzusehen. Damit ist die Software ein leistungsfähiger Ersatz für die üblichen Simulatoren. Einzig eine Demonstrationsmöglichkeit der Abfolge der Mikroschritte innerhalb eines Befehls vermisst man ein wenig.

Die Verwendung eines Mikrocontrollers soll nicht nur der Demonstration der Praxistauglichkeit der ebenfalls in anderen Lehrgängen besprochenen Inhalte dienen. Es ist zwar durchaus möglich, nur die typischen Aufgabenstellungen ohne dynamische Ein- und Ausgaben, wie sie von Schülern auf Simulatoren umgesetzt werden können, auf dem MSP430-Befehlssatz zu implementieren, jedoch bietet diese Methode kaum Vorteile gegenüber den Simulatoren. Der Mehraufwand, die deutlich kompliziertere, da nicht für Schulungszwecke ausgelegte Software zu nutzen, ist dafür wohl übertrieben. Die Verwendung eines Mikrocontrollers mit mehreren digitalen I/O-Ports ermöglicht jedoch die einfache Ansteuerung zusätzlicher Hardware. Daher bot sich die Entwicklung einer zusätzlichen Platine an, welche über acht einzeln ansteuerbare LEDs als Ausgabe sowie vier Taster als Eingabe verfügt (siehe 2.3.3).

2.3.1 Die MSP430-Architektur

Der MSP430 ist ein weit verbreiteter Mikrocontroller der Firma Texas Instruments. Es gibt eine Vielzahl von verschiedenen Varianten des Chips, sowohl in SMD- als auch in klassischer Bauweise. Dabei beziehen sich die Unterschiede im Wesentlichen auf die im Controller verbaute Peripherie wie Speichergröße, I/O-Ports, A/D-Wandler und ähnliches. Grundarchitektur und Befehlssatz der eigentlichen CPU sind jedoch für alle Versionen identisch.

Der MSP430 nutzt als klassischer von-Neumann-Rechner einen gemeinsamen Speicheraddressraum, welcher sowohl den Flash-Speicher, den Arbeitsspeicher sowie die Register für die verbauten Peripheriegeräte abdeckt. Die CPU verfügt, abgesehen von Program Counter (PC), Stack Pointer (SP), dem Status-Register (SR) und einem Constant Generator (CG) über 12 weitere, frei nutzbare Prozessorregister. Der RISC-Befehlssatz der CPU ist mit insgesamt 51 Befehlen sehr überschaubar. Obwohl der MSP430 als 16-Bit-Architektur konzipiert ist, ist der Speicher byteweise adressierbar und die Befehle stehen auch jeweils in einer 8-Bit-Variante zur Verfügung, erkennbar am jeweils angehängten `.b` (Byte).

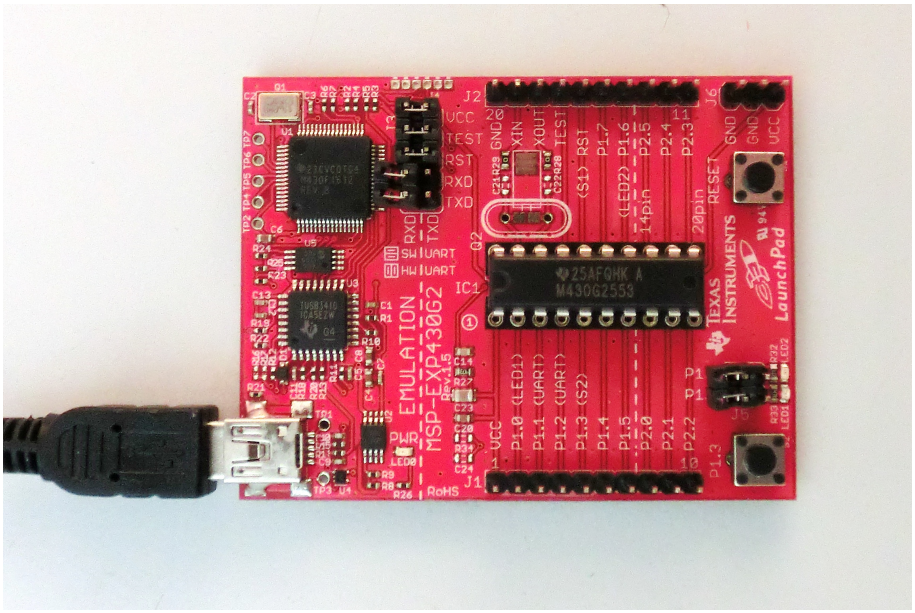

Abbildung 1: TI Launchpad

2.3.2 MSP430 Launchpad und das Code Composer Studio

Während dieser Mikrocontroller im professionellen Umfeld schon etliche Jahre erfolgreich eingesetzt wird, war die Verbreitung unter privaten Bastlern lange Zeit relativ gering. Erst mit Einführung des als Launchpad bezeichneten USB-Programmers wurde die Architektur auch für sie interessant. Das Launchpad (siehe Abb. 1) wird sowohl von TI direkt sowie über Zwischenhändler für etwa 5 Euro vertrieben. Es wird zusammen mit zwei damit programmierbaren MSP430 geliefert.

Das Launchpad soll interessierten Entwicklern einen schnellen Einstieg ermöglichen und kommt bereits ab Werk mit einem frei programmierbaren Taster sowie zwei ansteuerbaren Leuchtdioden. Es ist jedoch auch sehr einfach um ansteuerbare Hardware zu erweitern, da alle Anschlussbeinchen des Controllers über Stiftleisten nach außen geführt werden. Zusätzliche Hardware kann demnach einfach aufgesteckt werden (siehe 2.3.3).
Als Software zur Programmierung stehen auch einige freie Umgebungen zur Verfügung, TI selbst bietet ihre Entwicklungsumgebung “Code Composer Studio” für den Einsatz mit dem Launchpad gratis an.

Das Code Composer Studio (CCS), eine Programmierumgebung auf Basis des bekannten Eclipse, bietet insbesondere auch einen Debug-Modus

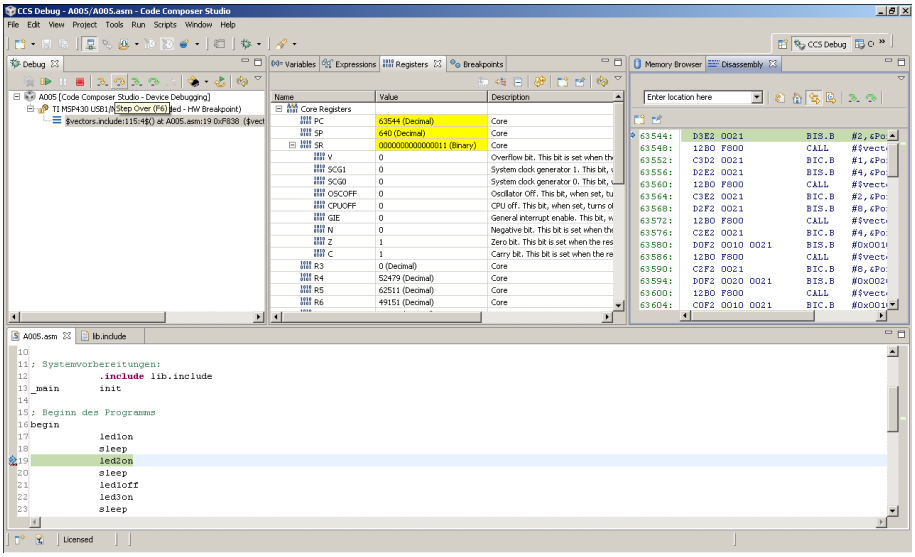

Abbildung 2: Debug-Modus des Code Composer Studios

(siehe Abb. 2) an. Dabei kann ein auf dem auf das Launchpad aufgesteckten MSP430 in Ausführung befindliches Programm genau analysiert werden. Erwartungsgemäß ermöglicht es der Debugmodus nicht nur, das Programm zu pausieren und in Einzelschrittfolge auszuführen, sondern gestattet nahezu grenzenlosen Einblick und Zugriffsmöglichkeiten in Prozessorregister und Speicher (siehe Abb. 3). Da die Software jedoch nicht für Einsteiger oder für Schulungszwecke konzipiert wurde, ist eine umfangreiche Einarbeitung, zumindest für den Unterrichtenden, unabdingbar. Beispielsweise bietet das CCS die Möglichkeit, die im Speicher bzw. in Registern abgelegten Werte auch in Dezimalschreibweise anzugeben. Nach der Installation wird es jedoch zuerst das Hexadezimalformat wählen. Jedoch sollte jedem, der etwas Erfahrung mit Eclipse mitbringt, der Einstieg schnell gelingen. Einige Hinweise zur Installation sind in Kapitel 3 gegeben.

2.3.3 Ein- / Ausgabehardware

Um anschauliche Programminteraktion zu ermöglichen, wurde auch eine Hardwareplatine (siehe Abb. 4) entwickelt. Damit ein hoher Grad an Transparenz für die Schüler erzeugt werden kann, wurden nur einfachste Bauteile verwendet und auf komplizierte Sensoren verzichtet. Das endgültige Layout umfasst acht Leuchtdioden, welche nur über Vorwiderstände direkt am ersten I/O-Controller des MSP430 angeschlossen sind, sowie vier einfache Taster an Port 2. Dies ermöglicht es, Taster und LEDs direkt über Zugriff auf die

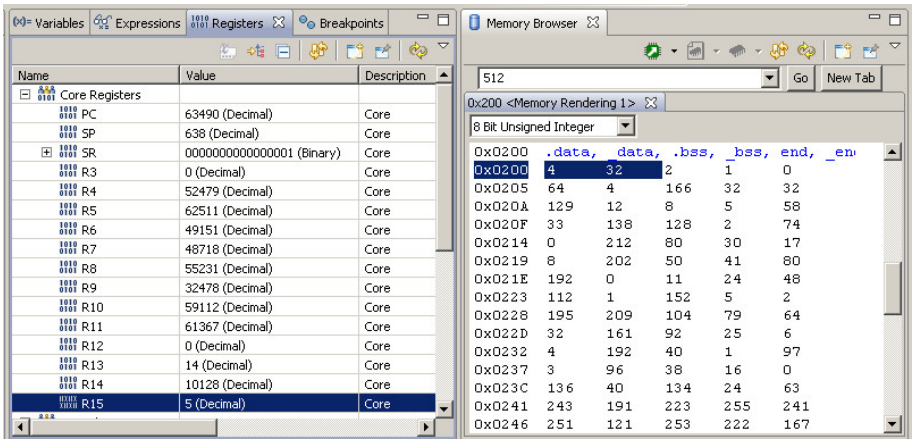

Abbildung 3: Register- und Memory-Browser des Code Composer Studios

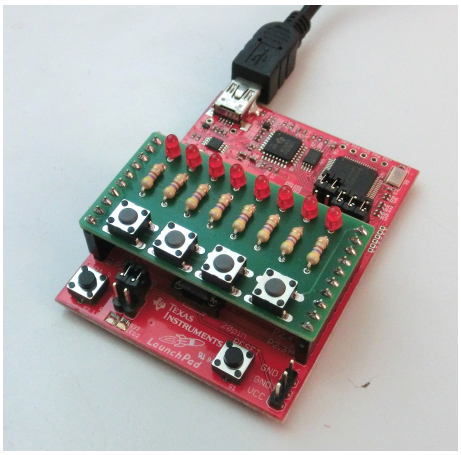

Abbildung 4: TI Launchpad mit E/A-Platine

entsprechenden Speicheradressen anzusprechen und somit direkt einen ein Byte großen Zahlenwert als Binärwert auf den LEDs auszugeben.

Die Platine hat inklusive der Bauteile Materialkosten von etwa 5 Euro, wobei ein Großteil der Kosten für die Herstellung der unbestückten Leiterplatte selbst zu kalkulieren ist.

Der Schaltplan der Platine befindet sich im Anhang auf Seite 28.

3 Durchführung

Die nachfolgende Durchführungsbeschreibung zielt darauf ab eine Grundlage zur Unterrichtsgestaltung zu sein, ist jedoch aufgrund äußerer Umstände für das Doppelstundenkonzept konzipiert. Es sollte jedoch mit geringem Aufwand auch auf dreistündige Kurse übertragbar sein. Dabei sieht jede Doppelstunde zuerst einen Theorieteil von ca. 30 Minuten vor, der von einem aufgabengeleiteten Praxisteil gefolgt wird. Dabei sind die Aufgaben nach Wichtigkeit und Schwierigkeit sortiert. Somit haben alle Schüler die Chance, die zum Verständnis unabdingbar wichtigen Aufgaben auszuführen, während begabteren Schülern die Möglichkeit gegeben wird, darüber hinaus das Erlernte weiter zu vertiefen.
Theorie- und Praxisphase sind im Verlauf der Sequenz, insbesondere gegen Ende, fachlich nicht zwingend ineinandergreifend, da einige der zu besprechenden Konzepte sich nicht direkt in Aufgaben widerspiegeln lassen. Am Ende der Stunde wird jeweils ein Fazit gezogen, wobei die Lösungen der gestellten Aufgaben kurz besprochen werden.

Die Aufgaben sollen von den Schülern eigenverantwortlich in Einzel- oder Partnerarbeit bearbeitet werden. Wie oftmals beim Einstieg in neue Programmiersprachen bzw. -umgebungen ist der erste Schritt der schwierigste. Sowohl das Code Composer Studio als auch der MSP430 selbst benötigen einige konkrete Einstellungen, welche für Einsteiger schwer verständlich sind. Um diese zur Programmierung notwendigen, aber für das Verständnis der Arbeitsweise überflüssigen "Hausmeisterarbeiten", beispielsweise die Deaktivierung des Watchdogs[5] und die Konfiguration der beiden I/O-Ports[6], elegant zu umgehen sowie weitere, nicht zum Befehlsumfang gehörende Makros zu definieren, sollen die Schüler für die Aufgaben entsprechende Vorlagen in das Code Composer Studio importieren (siehe Anhang Seite 30). Die Makros stellen insbesondere eine Möglichkeit dar, das aktuelle Programm zu beenden (hold), kurz zu pausieren (sleep) sowie, für die späteren Aufgaben notwendig, den Zustand der Taster abzufragen bzw. den Zustand der LEDs einzeln zu setzen.

[5]Der Watchdog ist ein Bestandteil des Mikrocontrollers, der den reibungslosen Ablauf überwachen soll, indem sich nach dem Prinzip des Totmannknopfes die CPU in regelmäßigen Abständen beim Watchdog melden muss um nicht neu gestartet zu werden. Damit sollen im Praxiseinsatz Abstürze und Ähnliches vermieden werden. Der Einsatz desselben führt aber ohne eingehendes Verständnis seiner Funktion zu intransparentem Verhalten des Controllers.

[6]Die Pins beider Ports des MSP430 können sowohl zur Ein- als auch zur Ausgabe genutzt werden. Dieses Verhalten kann in entsprechenden Speicherzellen konfiguriert werden.

Ablauf

1	Einstieg: von-Neumann-Architektur, Einführung in die Hard- und Software, Aufgaben zum Binärsystem
2	Speicherzugriffe und bedingte Sprünge: Aufbau des Speichers, Ablauf einer Sequenz von Befehlen, endlose Schleife
3	Arithmetische Operationen: Einführung Rechenwerk, Additions- und Subtraktionsbefehle
4	Bedingte Sprünge: Statusregister, Statusflags, bedingte Sprungbefehle, einfache Kontrollstrukturen
5	Schleifen und Bedingungen: Vertiefung Kontrollstrukturen, Ablauf, Zustandstabelle
6	Maschienensprache und Mikroschritte: Detaillierte Betrachtung des Ablaufs, Vergleich Hochsprache-Assemblersprache-Maschinensprache
7	Registermaschine: Einführung Registermaschinen, Vergleich Registermaschinene zu MSP430
8	Abituraufgaben

Vorbereitung

Um in der ersten Unterrichtsstunde ohne Umschweife beginnen zu können, sind einige Vorbereitungen nötig. Diese Arbeiten sind selbstverständlich, wenn die verwendeten Arbeitsplätze fest sind und durch die Schule verwaltet werden. Werden hingegen von Schülern selbst verwaltete Notebooks eingesetzt, muss die Software durch die Schüler vor Beginn des Projektes selbst installiert und getestet werden. Dabei sind einige Besonderheiten zu beachten.

Das “Code Composer Studio” ist die von TI selbst vermarktete Entwicklungsumgebung, welche zur Programmierung einer ganzen Reihe von Mikrocontrollern der Firma Texas Instruments konzipiert wurde. Die im Normalfall fälligen Lizenzkosten entfallen bei der sogenannten “Code Free Licence”, welche explizit für die Verwendung mit dem Launchpad zur Verfügung steht. Die dadurch entstehende Einschränkung im Funktionsumfang, eine Beschränkung der Programmgröße, spielt beim Einsatz mit dem Launchpad keine Rolle, da die mitgelieferten Chips gar nicht mehr Speicherplatz zur Verfügung stellen.

Bei der Installation sind insbesondere noch die zur Übertragung wichtigen Treiber und Entwicklungswerkzeuge auszuwählen, namentlich “Compiler Tools”, “Device Software” und “JTAG Emulator Support” sowie die Unterstützung der MSP430-Prozessoren.

Auch am Launchpad sollte einiges beachtet werden: Da die direkt auf dem Launchpad angebrachten LEDs und Taster nicht benötigt werden, sind die entsprechenden Jumper zu öffnen. Auch die Verbindung die mit "RXD" und "TXD"[7] beschrifteten Jumper sollten geöffnet werden, da ansonsten, je nach Status der LEDs, das Launchpad vom Treiber nicht korrekt erkannt wird. Alle sonstigen Steckbrücken müssen geschlossen bleiben (siehe Abb. 1).

Sollte alles korrekt installiert worden sein, wird ein neu angeschlossenes Launchpad von Windows nun korrekt erkannt und die erforderlichen Treiber aktiviert. Zum Test der Programmierumgebung sowie der Debugschnittstelle wurde ein Demonstrationsprojekt vorbereitet, welches im Debug-Modus gestartet wird.

Die Installationen der Schüler verliefen in den meisten Fällen fehlerfrei, in wenigen Fällen mussten an den Konfigurationen im Nachhinein noch Änderungen durchgeführt werden, welche sich aber durch genaues Lesen und Befolgen der verteilten Installationsanleitung erübrigt hätten. Einzig ein Notebook lies sich gar nicht zur Zusammenarbeit mit dem Launchpad bewegen. Eine genauere Fehlersuche zeigte, dass falsche Mainboardtreiber, bzw. die zum Mainboard gehörenden USB-Host-Treiber, die Ursache dafür waren.

Das Programm funktionierte auch in einer virtuellen Umgebung - ein Windows XP in Virtualbox - hervorragend. Somit ist es auch denkbar, korrekt installierte virtuelle Maschinen zu verteilen.

3.1 Einstieg

Der Handreichung des ISB(4) folgend wird zum Einstieg in das Thema gemeinsam mit den Schülern ein alter Arbeitsplatzrechner zerlegt. Dabei werden die einzelnen Bestandteile mit den Schülern besprochen. Die an der Tafel vermerkten Komponenten werden anschließend grob in Zentraleinheit und Peripheriegeräte gegliedert. Eine genauere Betrachtung der internen Komponenten führt dann zu der Gliederung nach von Neumann, wobei mit den Schülern die einzelnen Komponenten dem vorgestellten Schemasystem zugewiesen werden.

Anschließend werden die Mikrocontroller verteilt. Die Schüler versuchen zu erraten, welche der von-Neumann-Komponenten sie nun in der Hand halten. Die überraschende Lösung: sämtliche Komponenten, also ein komplett funktionsfähigen Rechner, ist in nur einem Chip verbaut. Um die nun aufgebaute Motivation noch zu steigern, werden einige Minuten zur Diskussion

[7] Der MSP430 verfügt über UART-Funktionalität (Universal Asynchronous Receiver Transmitter). Diese Funktionen sind jedoch an PINs verfügbar, welche in diesem Fall LEDs ansteuern sollen. Um Probleme zu vermeiden wird durch Trennung der Verbindung zum Launchpad der UART außer Kraft gesetzt.

der Verbreitung und Nutzung von solchen Mikrocontrollern genutzt.

Nun wird die verwendete Hard- und Software ausführlich vorgestellt. Dabei muss auch auf eine saubere Trennung der Funktionen des Launchpads, des MSP430 und der aufgesteckten Platine geachtet werden. So ist klarzustellen, dass das eigentliche System, der MSP430, nur aus einem einzigen, auf das Launchpad aufgesteckten Chip besteht. Die darüber aufzusteckende Platine mit den 8 LEDs und 4 Tastern dient nur zur Ein- und Ausgabe, gewissermaßen als einfacherer Ersatz für Monitor und Tastatur bei normalen PCs. Das Launchpad dient im wesentlichen nur der Programmierung des MSP430 über USB, wobei es ebenfalls eine detaillierte Einsicht in das in Ausführung befindliche Programm erlaubt. Letztlich ließe sich aber der MSP430 auch ohne Launchpad betreiben.

Der erste Praxisteil dient mehrern Zielen. Einerseits sollen die Schüler sich mit der Programmierumgebung und dem Launchpad vertraut machen, andererseits entwickeln sie, durch einfache Aufgaben geleitet, selbst ein Verständnis für das Binärsystem. Dazu importieren die Schüler zu Beginn ein lauffähiges Projekt, welches einige Reihe an LEDs zum leuchten bringt. Das Programm wird auf den MSP430 übertragen, indem der Debug-Modus gestartet wird. Dort kann auch der Programmablauf direkt beobachtet werden.

Aufgabe: Ansteuerung der LEDs
Importieren Sie zunächst die Projektdatei Aufgabe01.zip in das Code Composer Studio.

Das funktionsfähige Projekt bringt einige LEDs zum leuchten. Dafür ist im Wesentlichen der Befehl

```
mov.b #52, &33
```

verantwortlich, der den Wert 52, als Zahlenwert gekennzeichnet durch #, an die Speicheradresse 33, als Adresse gekennzeichnet durch &, ablegt. An dieser Speicheradresse „lauscht“ der Output-Conroller, welcher die LEDs schaltet.

Aufgabe ist nun, herauszufinden, nach welchem System die LEDs zu den Zahlenwerten leuchten.

1. Testen Sie zuerst einige weitere ein- bis zweistellige Zahlenwerte und beobachten Sie, was passiert. Versuchen Sie es insbesondere auch mit dem Wert #0.
2. Versuchen Sie anschließend gezielt, die LEDs jeweils einzeln zum Leuchten zu bringen. Schreiben Sie sich die jeweils zu den LED gehörenden Werte auf.

3. Versuchen Sie nun zuerst vorherzusagen, was bei folgenden Werten passiert. Testen Sie danach, ob ihre Vorhersage stimmt:
   (a) #32
   (b) #33
   (c) #3
   (d) #6
   (e) #7
   (f) #128
   (g) #127
   (h) #255
4. Sie sollten nun in der Lage sein folgende LED-Muster zu erzeugen: (X an, 0 aus)
   (a) 00000X0X
   (b) 0000X00X
   (c) X00XX00X
   (d) XXXX0000
   (e) XX000000
   (f) 000X0XXX
   (g) 00X0X0X0
5. *Zusatzaufgabe: Erstellen Sie ein Lauflicht!*

Das ursprüngliche Projekt sieht nur einen einzigen Zahlenwert vor, welcher im Speicher abgelegt wird und anschließend keine weiteren Befehle mehr ausführt. Diese Vorgehensweise erweist sich schnell als ineffizient, da der Startvorgang nach jeder minimalen Änderung des Programms etliche Sekunden in Anspruch nimmt. So bietet es sich an, mehrere Befehle untereinander zu reihen. Damit die Schüler die Befehle dann einzeln ausführen können, wird ihnen bereits hier die Möglichkeit der einzelschrittweisen Ausführung im Debug-Modus vorgestellt.

Im Abschluss der Stunde werden die Lösungen der Aufgaben kurz besprochen und dabei auf das Binärsystem eingegangen.

3.2 Speicherzugriffe und unbedingte Sprünge

Der Wiederholung des Binärsystems zu Beginn der zweiten Doppelstunde folgt eine kurze Erwähnung des Hexadezimalsystems. Das Ziel ist jedoch

nur, die Schüler darauf vorzubereiten, dass das Code Composer Studio teilweise Zahlenwerte in diesem System anzeigen wird. Um keine weitere Zeit mit überflüssigen Rechenübungen zum Binärsystem zu verbringen wird den Schülern nahegelegt im Zweifelsfall den Windows-Taschenrechner zur Umrechnung zu nutzen.

Die Kerninhalte der zweiten Stunde sind einerseits die intensive Besprechung des Speichers und die Einführung des Akkumulators als "Zwischenspeicher" und andererseits die detaillierte Besprechung der Abarbeitung einer linearen Befehlssequenz.

Als Einstieg wird nochmals der einzige bereits bekannte Befehl genauer beleuchtet: `mov`.Er verschiebt einen ganzzahligen Zahlenwert in eine adressierte Speicherzelle. Darauf aufbauend wird der Speicher an sich genauer beleuchtet. Der Adressraum des MSP430 im Besonderen wird kurz angesprochen und die gemeinsame Verwendung als Schnittstelle zu den Controllern, als Hauptspeicher und als Befehlsspeicher werden mithilfe des Memory Browsers demonstriert und als ein wesentliches Kriterium der von-Neumann-Architektur klargestellt.

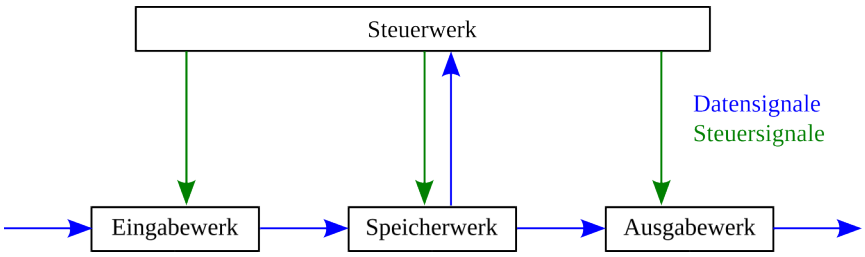

Abbildung 5: Tafelbild: Datenflüsse im von-Neumann-Prinzip (ohne Rechenwerk)

Um Werte aus einer Speicherzelle in eine andere zu kopieren, wird erstmals der Akkumulator angesprochen. Obwohl der MSP430 insgesamt elf funktionsgleiche Register besitzt, wird bereits hier festgelegt, dass im gesamten Unterrichtsverlauf nur ein einziges, das Register R15, für diese Funktion genutzt wird. Damit wird einerseits gezeigt, dass ein Akkumulatorregister für alle Operationen vollkommen ausreichend ist, andererseits erarbeiten sich die Schüler bereits die Algorithmen mit Zwischenspeicherungen im Hauptspeicher, welche für die späteren Aufgaben an den Registermaschinen unabdingbar sind.

Anschließend kann nun die Abarbeitung der Befehle genauer betrachtet werden. Dazu wird die "Disassembly"-Anzeige des CCS vorgestellt: hier lassen sich jeweils zu den Befehlen die Speicheradressen auslesen (siehe Abb.

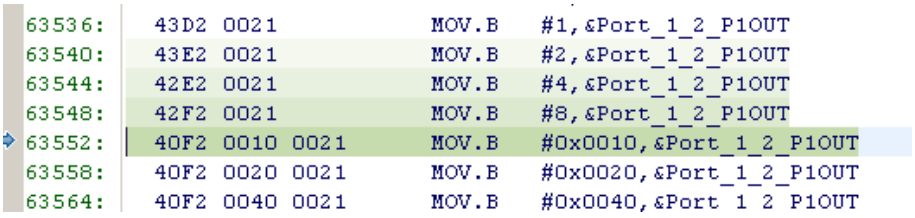

Abbildung 6: Disassembly-Ansicht des Code Composer Studios

6). Über die Schlüsselfrage, welche Informationen ausreichend sind, um den gesamten Zustand des Mikrocontrollers korrekt abzuspeichern, kommt man schnell zum Befehlszählerregister (Program Counter, PC). Hier lässt sich nun in Einzelschrittausführung gut demonstrieren, wie die Adresse des nächsten auszuführenden Befehls, sichtbar in der Disassembly-Anzeige, bereits im PC liegt.
Damit lässt sich bereits grob der grundlegende Abarbeitungsalgorithmus besprechen: Nach jedem bearbeiteten Befehl muss der PC erhöht werden, um anschließend den nächsten Befehl interpretieren zu können. Darauf aufbauend lässt sich nun eine Endlosschleife implementieren. Hier bietet sich die Möglichkeit an, zuerst durch einen `mov`-Befehl direkt in den PC die Adresse des nächsten Befehls zu schreiben. Damit lässt sich das Prinzip anschaulich demonstrieren. Da die Adressen jedoch bei der Entwicklung des Assemblercodes noch nicht bekannt sind, ist der Vorteil der Sprungbefehle durch Label, die genau in die demonstrierten Maschinenbefehle übersetzt werden, klar. Im nachfolgenden sollen die Schüler auch nur die Sprünge zu Labels mit dem Befehl `jmp` *`label`* nutzen.

Dieses Vorgehen und das Zusammenspiel wird an einer momentan noch stark vereinfachten systematischen Darstellung von Speicher und CPU mit PC und Akku erklärt (siehe Abb. 7). Dieses Abbild wird in den folgenden Stunden sukzessive erweitert, jedoch bleibt die Grundstruktur unverändert.

An dieser Stelle ist es überflüssig, auf die im realen Fall unterschiedliche Länge der Befehle einzugehen. Die Schüler können zwar sehen, dass die mov-Befehle jeweils mehrere Speicherzellen einnehmen, dieser Umstand kann aber mit dem Hinweis auf die technische Umsetzung des MSP430 erklärt werden.

Die nachfolgenden Aufgaben dienen der Vertiefung des Besprochenen. Insbesondere der Memory Browser soll dabei eingesetzt werden - einerseits um den Speicher bei den Operationen in Einzelschritten zu Überwachen, andererseits, um Speicherzellen direkt zu ändern.

Aufgabe: Speicherzugriffe

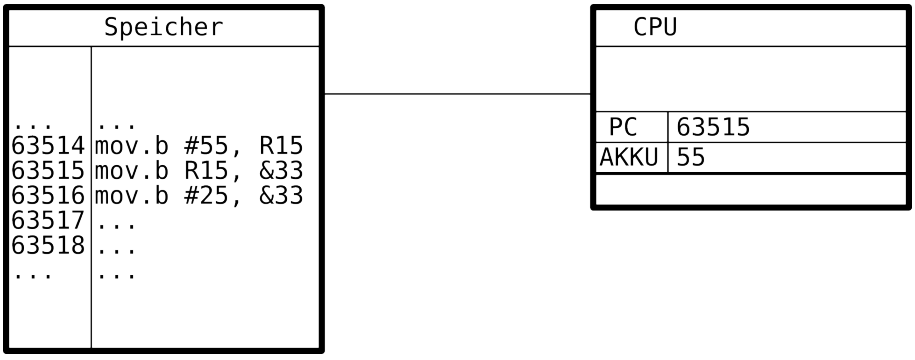

Abbildung 7: Tafelbild zur Besprechung des sequenziellen Ablaufs

Importieren Sie zunächst die Projektvorlage Aufgabe02.zip in das Code Composer Studio.

1. Benutzen Sie den Befehl `mov.b` um einige Werte aus dem Hauptspeicher (ab Adresse &512) über ein Akkumulatorregister (R15) an den LED-Controller (&33) zu kopieren, um die Werte aus dem Hauptspeicher anzuzeigen.
2. Eingaben sind in unseren Mikrocontrollern mittels der vier Taster möglich. Der durch Tastenkombinationen erzeugte Eingabewert kann über Speicheradresse &40 ausgelesen werden. Schreiben Sie ein kurzes Programm, welches den Wert liest und (über den Akkumulator!) an den Ausgabespeicher des LED-Controllers kopiert. Wenn sie dieses Codefragment in eine Schleife legen, sollten Sie die Möglichkeit haben, direkt mit den Tasten die LEDs zu steuern.
3. Zusatzaufgabe: Erstellen Sie ein Lauflicht oder andere Animationen auf den LEDs.

3.3 Arithmetische Operationen

Das bisher erabeitete Modell des von-Neumann-Rechners bietet keine Möglichkeit der Verarbeitung der Daten. Auch die bisher besprochenen Befehle des MSP430 bewirken nur Verschiebungen der Daten im Speicher. Das bereits bekannte, unvollständige Bild der Datenflüsse wird nun um das Rechenwerk erweitert. Dazu werden insgesamt vier Befehle vorgestellt: `inc.b`, `dec.b`, `add.b` und `sub.b`[8]. Während das Inkrementieren und Dekre-

[8]Hinweis: Der Befehlssatz des MSP430 bietet keine eigenen Operationen zur Multiplikation bzw. Division. Sollten diese jedoch als pseudoatomare Operation gebraucht werden,

mentieren jeweils nur das betroffene Register als Operand benötigt, sind die Befehle für Addition bzw. Subtraktion auf 2 Operanden angewiesen. Dabei sollen die Befehle jedoch wieder nur auf ein einziges Register angewendet werden (siehe 3.2)

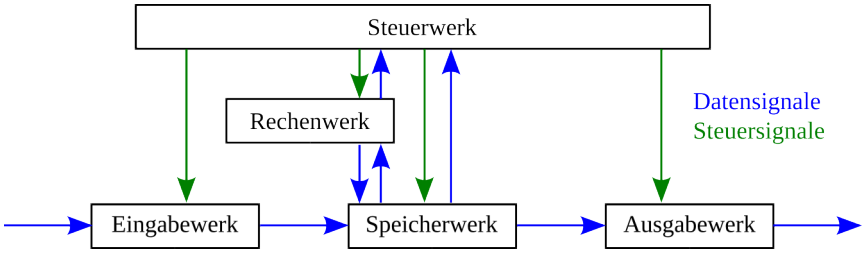

Abbildung 8: Tafelbild: Datenflüsse im von-Neumann-Prinzip

Auch das Tafelbild der letzten Stunde zum Zusammenspiel zwischen Speicher und Zentraleinheit wird nun wieder aufgegriffen und um das Rechenwerk mit seinem Zugriff auf den Akkumulator ergänzt. Mit dem Vorwissen der letzten Stunde zum Befehlsablauf lässt sich nun der Befehl `inc.b` und anschließend auch `add.b` durchspielen.

Aufgabe: Arithmetische Operationen
Importieren Sie zunächst die Projektvorlage Aufgabe03.zip in das Code Composer Studio.

1. Der Befehl `inc R15` erhöht den Wert im Akkumulatorregister R15 um eins. Schreiben Sie ein Programm, das aus dem Hauptspeicher (&512) einen Wert in den Akkumulator lädt, um eins erhöht, und wieder an der ursprünglichen Speicheradresse ablegt.
2. Was macht folgendes Programm?

ließen sich diese als Makros in den Vorlagen implementieren. Alle weiteren arithmetischen bzw. logischen Operationen des Prozessors, beispielsweise Bitrotationen und Maskierungen sind für den Schulunterricht eher ungeeignet.

```
; Beginn des Programms
       mov.b #0,&33
start
       mov.b &33, R15
       inc.b R15
       mov.b R15, &33
       sleep
       jmp start
; Ende des Programms
```

Versuchen Sie zuerst, das Programm zu verstehen und probieren Sie es danach aus!

3. Der Befehl `add.b` erlaubt es, beliebige Zahlen oder auch Speicherinhalte zu einem Akkumulatorregister zu addieren.

   Schreiben Sie ein Programm, das die Zahlen #34 und #19 addiert. Beobachten Sie den Programmablauf Schritt für Schritt im Register. Wo ist das Ergebnis?

   Ändern Sie das Programm ab, damit es die Inhalte der ersten beiden Speicherzellen des Arbeitsspeichers (&512 und &513) addiert und das Ergebnis an der Speicheradresse &514 ablegt. Beobachten Sie den Programmablauf Schritt für Schritt in Speicher und Register.

4. Der Befehl `sub.b` subtrahiert aus dem angegebenen Akkumulatorregister einen Wert oder Speicherinhalt. Entwickeln Sie ein Programm, welches von #100 eine beliebige Zahl kleiner 100 abzieht. Beobachten sie das Vorgehen genau im Akkumulatorregister. Ziehen Sie jetzt eine Zahl größer 100 ab. Wie interpretieren Sie das Ergebnis? Beobachten Sie auch die anderen Register genau bzgl. Veränderungen.

5. Versuchen Sie nun, einen einfachen Taschenrechner zu programmieren. Dieser soll nacheinander 2 mal Binärwerte über die 4 Knöpfe einlesen und addieren. Das Ergebnis soll dann auf den LEDs dargestellt werden. Das Programm kann jedoch leider nicht warten, bis Knöpfe gedrückt werden, daher ist es nur möglich, es durch Einzelschrittausführung (Step Over) auszuführen!

3.4 Bedingte Sprünge

Mit den bisher bekannten Sequenzen von Befehlen sowie endlosen Schleifen durch den unbedingten Sprungbefehl lassen sich selbst einfachste Algorithmen noch nicht umsetzten. Aus dieser Not heraus erkennt man die Notwendigkeit weiterer Befehle, um die aus Hochsprachen bekannten Kontrollstrukturen im Prozessor abzubilden. Der eindimensionale Speicherraum, in welchem die Befehle abgelegt werden müssen, lässt dabei schnell entwickeln, dass Verzweigungen im Programmablauf, wie sie bei einer bedingten Anweisung nötig sind, an verschiedenen Stellen im Befehlsspeicher abgelegt werden müssen. Es sind also weitere Sprungbefehle nötig, um diese Befehle im Speicher zu erreichen.

Durch einige ausgewählte Beispiele lassen sich im Code Composer Studio die drei zur Verfügung stehenden Statusflags zeigen. Dazu wird einfach in Einzelschrittausführung zunächst ein Überlauf im Speicher ausgelöst, um das Carry-Bit zu setzen. Dabei erkennt man in der Registeransicht nicht nur eine Veränderung des Akkumulatorregisters R15, sondern ebenfalls eine Änderung am als "SR" (Status Register) benannten Registers. Darin verbergen sich eben unter anderem auch die Statusbits der Sprungbefehle, welche in der Registeransicht - auf Englisch - ausreichend erklärt sind, um als solche erkannt zu werden (siehe Abb. 9). Anschließend lassen sich auf gleiche Weise das Zero-Bit sowie das Negative-Bit demonstrieren.

GIE	0	General interrupt enable. This bit, when set, enables maskable interrupts. When reset, all maskable interrupts are disabled.
N	0	Negative bit. This bit is set when the result of a byte or word operation is negative and cleared when the result is not negative.
Z	1	Zero bit. This bit is set when the result of a byte or word operation is 0 and cleared when the result is not 0.
C	1	Carry bit. This bit is set when the result of a byte or word operation produced a carry and cleared when no carry occurred.
R3	0 (...	Core

Abbildung 9: Status-Flags in der Register-Ansicht des Code Composer Studios

Nach kurzer Vorstellung der aus den drei Statusflags resultierenden Sprungbefehlen (`JC, JZ, JN`) und deren Negationen (`JNC, JNZ`)[9] wird zunächst die Umsetzung der einseitigen Bedingung gemeinsam besprochen. Die weiteren Möglichkeiten der bedingten Sprünge werden in den darauf folgenden Aufgaben erarbeitet.

> **Aufgabe: Bedingte Sprungbefehle**
> Importieren Sie zunächst die Projektvorlage Aufgabe04.zip in das Code Composer Studio.

[9]Hinweis: Der MSP430 bietet keinen Sprung bei explizit ungesetztem Negative-Bit an. Wird dieser gebraucht, kann er natürlich ebenfalls als Makro implementiert werden.

1. Zweiseitige Bedingung
   Nutzen Sie den Befehl testBtn1, um zu erkennen, wenn der Taster 1 gedrückt wird. Ist er gedrückt, wird der Befehl das Zero-Bit setzen. Konstruieren Sie damit eine zweiseitige Abfrage, um bei Knopfdruck alle LEDs einzuschalten. Wird der Knopf losgelassen, sollten alle LEDs wieder ausgehen.

2. Einseitige Bedingung
   Verändern Sie ihr Programm nun wie folgt: Zu Programmstart sollen alle LED aus sein. Wird der erste Taster gedrückt, sollen alle LEDs angehen und auch eingeschaltet bleiben, wenn der Knopf losgelassen wird.

   Können Sie noch einbauen, dass Knopf 2 die LEDs wieder ausschaltet?

3. Schleife mit Eingangsbedingung
   Was macht folgendes Programm?

```
mov.b #4, R15
begin
dec.b R15
jz end
mov.b #255, &33
sleep
mov.b #0, &33
sleep
jmp begin
end mov.b #0, &33
hold
```

   Versuchen Sie es zuerst, es zu verstehen. Führen Sie es dann aus, um Ihre Vermutung zu prüfen.

   Modifizieren Sie dann das Programm wie folgt: Anstatt herunterzuzählen (dec) können Sie, ohne die offensichtliche Funktion des Programms zu verändern, auch beginnend mit #0 hinaufzählen (inc.b).

3.5 Schleifen und Bedingungen

Die bereits bekannten Sprunganweisungen sollen nun vertieft werden. Dazu werden zuerst exemplarisch die Assemblercodes zu zwei in Pseudocode bzw. Hochsprache geschriebenen Kontrollstrukturen erarbeitet (siehe Abb. 10). Diese werden anschließend im um die Statusflags erweiterten Tafelbild (siehe Abb. 11) nochmals ausführlich besprochen, um nochmals den Ablauf intensiv

einzuüben.
Die Funktion des Carry-Bits wird den Schülern nicht über Aufgaben nähergebracht, sondern anhand der Addition zweier 16-Bit-Werte mittels der 8-Bit-Addition (`add.b`) demonstriert.

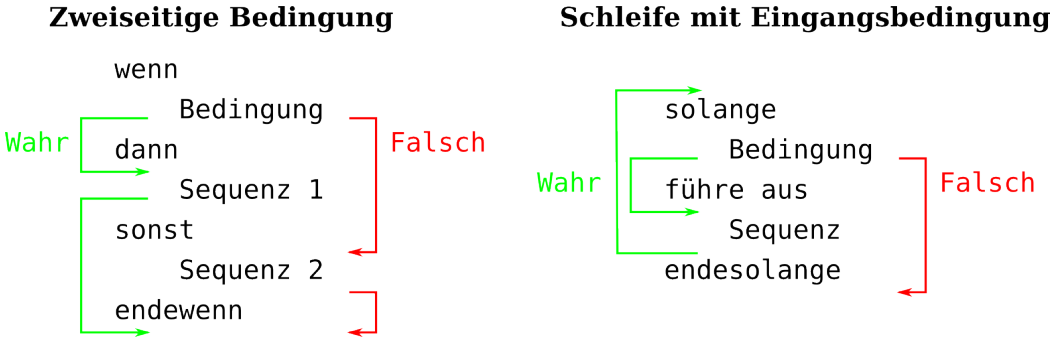

Abbildung 10: Tafelanschrieb: Bedingte Sprünge, Kontrollstrukturen

Als Hilfsmittel zur Überprüfung der geschriebenen Algorithmen wird nun auch die Zustandstabelle eingeführt. Anhand einiger kurzer Beispiele werden dabei auch bei unbekanntem Code schnell die zugrundeliegenden Strukturen erkannt.

Die weiteren Aufgaben sollen das bisher einstudierte weiter vertiefen.

Aufgabe: Bedingte Sprungbefehle
Importieren Sie zunächst die Projektvorlage Aufgabe04.zip in das Code Composer Studio.
Die folgenden Aufgaben erfordern viele Befehle, dabei müssen

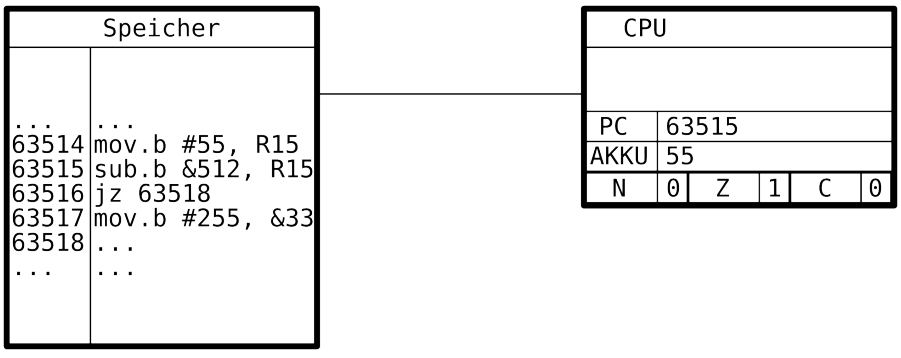

Abbildung 11: Tafelbild zur Besprechung des Ablaufes einiger Algorithmen (Beispiel)

Sie oftmals Rechenzwischenschritte für spätere Verwendung zwischenspeichern. Benutzen Sie dazu die Speicherzellen des Hauptspeichers (&512 ff.). Gehen Sie bei der Programmierung kleinschrittig vor und testen Sie ihre Zwischenlösungen mittels Einzelschrittausführung.
Die Aufgaben sind nicht aufeinander aufgebaut. Suchen Sie sich heraus, welche Sie zuerst lösen möchten!

1. Multiplikation
   Der MSP430 hat keinen eingebauten Befehl für die Multiplikation. Jedoch ist die Multiplikation mit einer Zahl n nichts anderes als eine n-Fache Summe über einen Wert. (z.B. 5x3 = 3+3+3+3+3) Schreiben sie ein Programm, das die Werte der Speicheradressen &512 und &513 multipliziert. Legen Sie dazu ihre Zwischensumme immer in &514 ab.
2. Umständliche Tastatur
   Zu Beginn sollen alle LEDs aus sein (Wert 0). Durch Drücken des ersten Knopfes soll der Wert um 1 erhöht werden. Durch Drücken des rechten Knopfes soll der Wert wieder um 1 gesenkt werden. Der vierte Knopf soll den Speicher wieder zurücksetzten.

   Für Experten: Erweitern Sie Ihr Programm zu einem einfachen Taschenrechner: Der dritte Knopf soll den momentan eingegebenen Wert zu den bereits vorher eingegebenen addieren (legen Sie die Zwischensummen in Speicher &512 ab). Der vierte Knopf soll nun nicht zurücksetzen, sondern die bisherige Lösung anzeigen.
3. Verschachtelte Schleifen
   Lassen Sie die LEDs blinken: Zuerst die erste LED einmal, dann die zweite LED zweimal, dann die LED-Kombination für den Wert dreimal u.s.w.

3.6 Maschinensprache und Mikroschritte

Die Theoriephase zu Beginn der sechsten Doppelstunde widmet sich nun der Maschinensprache, Assemblersprache und Hochsprache sowie der automatischen Übersetzbarkeit zwischen diesen. Leider zeigt sich hier der MSP430 als etwas zu kompliziert für die ausführliche Besprechung, da zu jedem

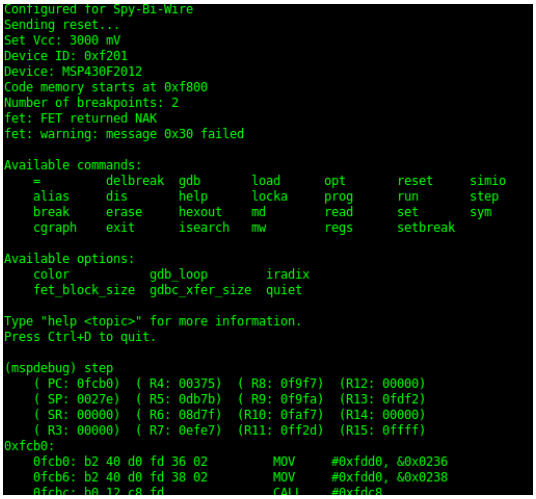

Abbildung 12: Software mspdebug

Assemblerbefehl nicht nur ein eindeutiger Maschinensprachebefehl existiert, sondern je nach Art der Operanden unterschieden werden muss. Daher wird hier bewusst nur das Prinzip erklärt und mit offenkundig nur beispielhaften Maschinencodes an der Tafel besprochen.

Zur Demonstration der prinzipbedingten Möglichkeit der Rückumwandlung von Maschinensprache in Assembler wird ausnahmsweise ein anderes Werkzeug eingesetzt: die freie Software mspdebug[10] (siehe Abb. 12). Diese Terminalanwendung erlaubt es einen momentan mit beliebigem Code bespielten MSP430 der Schüler auszulesen und sowohl in Maschinensprache als auch in einer disassemblierten Variante vorzustellen. Dieser Aspekt fehlt leider im CCS, es kann nur ein dort geschriebener Quelltext im Debugger inspiziert werden.

Anschließend kann im Code Composer Studio noch anschaulich gezeigt werden, wie ein Algorithmus in Hochsprache ebenfalls auf dem MSP430 ausgeführt wird. In der Analyse im Debugmodus zeigt sich, dass natürlich auch dieser in Maschinensprache umgewandelt wurde. Auch die prinzipiell schwierige Rückumwandlung in eine Hochsprache wird angesprochen. Die bei der Programmierung von Mikrocontrollern übliche Sprache C sollte aufgrund der syntaktischen Ähnlichkeit zu Java den Schülern beim Lesen keine Probleme bereiten.

Das zweite Unterrichtsziel dieser Unterrichtsstunde ist die Konkretisierung des Ablaufs eines einzelnen Befehls. Aus den Vorstun-

[10] http://mspdebug.sourceforge.net/

den ist bereits bekannt, dass jeweils der Befehl an der im PC abgelegten Adresse als nächstes ausgeführt wird und anschließend der PC inkrementiert wird. Dieser grundlegende Ablauf wird nun auf die bekannte von-Neumann-Befehlsfolge erweitert: Fetch, Decode, Fetch Operands, Execute, Update Program Counter.
Um diese Abfolge nun auch detailiert durchzuspielen, wird das bereits bekannte Tafelbild (siehe Abb. 7 bzw. Abb. 11) um das Befehlsregister (Instruction Register) erweitert.

Die Schüler kennen nun sämtliche Befehle und Vorgehensweisen und können damit auch komplexere Probleme zu lösen. Daher werden nun für diese und die darauffolgende Stunde nur Ideen für kleine Projekte geliefert, aus denen ausgewählt werden kann. Die Schüler können auch eigene Ideen verwirklichen.

Aufgabe: Anwendung
Importieren Sie zunächst die Projektvorlage Aufgabe05.zip in das Code Composer Studio.

Sie beherrschen nun ausreichend Befehle, um komplexere Programme zu entwickeln. Überlegen Sie sich selbst, was Sie gerne umsetzen möchten. Fragen Sie jedoch besser vorher ab, ob Ihre Idee in der vorgegebenen Zeit verwirklicht werden kann!

Beispiele:

- Codeschloss: Nur bei Eingabe der richtigen Reihenfolge der Tasten geht die Alarmanlage (die LEDs) aus.
- Elektronischer Würfel: Bei Druck auf eine Taste wird eine Zufallszahl (1-6) auf den LEDs ausgegeben. (Hinweis: Um Zufallszahlen zu erzeugen, können Sie eine Inkrementierung des Registers in eine Schleife setzten, welche bei Tastendruck unterbrochen wird.)
- Reaktionstester: Bei Aufleuchten der LEDs soll möglichst schnell die richtige Taste gedrückt werden.
- PWM (Pulsweitenmodulation): Sie können die LED “dimmen”, indem Sie diese abwechselnd Ein- und Ausschalten. Durch Modulation der Einschaltzeit gegenüber der Auszeit kann die Helligkeit beeinflusst werden. Implementieren Sie eine Steuermöglichkeit über die Taster!

3.7 Registermaschinen

In dieser Doppelstunde wird der bisher übliche Unterrichtsablauf ausnahmsweise umgekehrt: Die Praxisphase am MSP430 geht diesmal der theoretischen Besprechung voraus, um den in der letzten Stunde begonnenen Projekten ausreichend Zeit zur Vollendung einzuräumen.

Da nun der reale Prozessor und die damit möglichen Algorithmen ausreichend behandelt wurden, ist es nun an der Zeit, den wichtigen Rückschluss auf die theoretischen Registermaschienen zu vollenden. Dazu werden gemeinsam anhand einiger Beschreibungen von Registermaschinenen und der Dokumentationen der üblicherweise im Unterricht eingesetzten Simulatoren die Gemeinsamkeiten und Unterschiede zur MSP430-Hardware herausgestellt. Dabei wird insbesondere auf Aufbau, Speicher- und Bussystem und Befehlssyntax eingegangen. Abgesehen von den konsequent deutschen Bezeichnungen der Register stellt nur der Umstieg auf den einfacheren Ein-Adress-Befehlssatz die Schüler vor eine kleine Transferleistung.

3.8 Abituraufgaben

In der letzten Unterrichtsstunde vor der Klausur wird die Funktionsweise der Registermaschinen nochmals vertieft und anschließend anhand der zum Thema passenden Aufgaben der bisherigen Abiturprüfungen intensiv einstudiert. Dabei sind von den Schülern bereits Abituraufgaben vorbereitet worden.

Die Aufgaben aus dem Probeabitur bleiben unbesprochen, um den Schülern nochmals die Chance zu geben, sich auf die Klausur selbstständig vorzubereiten. Die Musterlösungen dafür sind verfügbar.

Die Klausur in der darauf folgenden Stunde stellt den Abschluss der Unterrichtseinheit dar und orientiert sich sowohl im Hinblick auf das Niveau als auch auf den Aufgabenstil sehr an den Abituraufgaben, ist jedoch etwa doppelt so umfangreich.

4 Anhang

Hardwareplatine

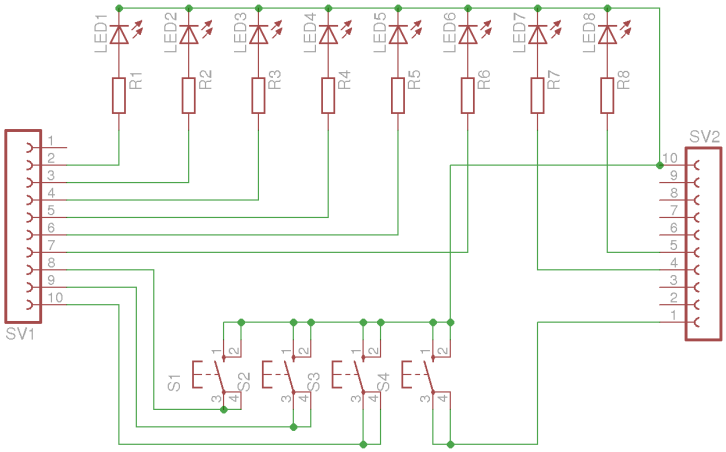

Abbildung 13: Schaltplan der entwickelten Erweiterungsplatine

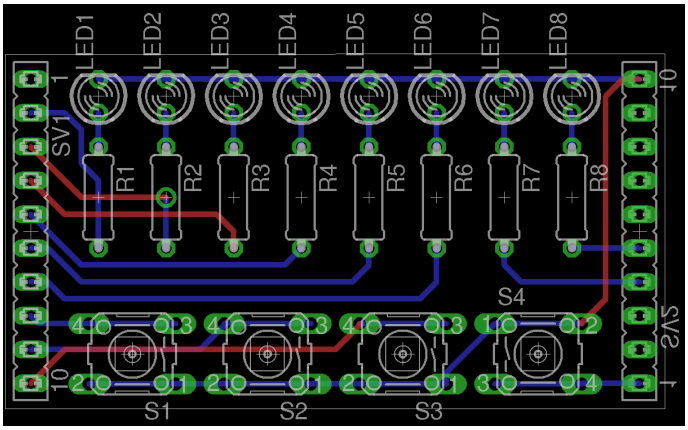

Abbildung 14: Layout der entwickelten Erweiterungsplatine

Befehlsübersicht

Die nachfolgende Tabelle zeigt alle den Schülern vorgestellten Befehle sowie die in den Vorlagen zur Verfügung gestellten Makros. Der Quelltext der Makros befindet im Anhang auf Seite 30 sowie auf der beigefügten CD.

Befehl	**Parameter**	**Erklärung**
mov.b	von, nach	Kopiert ("move") einen Wert (#) oder den Inhalt einer Speicherzelle (&) oder eines Registers (R) in eine andere Speicherzelle bzw. ein Register (*Bsp:* `mov.b #24, &33` *oder* `mov.b &40, R15`)
add.b	Wert, Reg	Addiert einen Wert (#) oder den Inhalt einer Speicherzelle (&) oder eines Registers (R) zu einem Register dazu (*Bsp:* `add.b #24, R15` *oder* `add.b &512, R15`)
sub.b	Wert, Reg	Subtrahiert einen Wert (#) oder den Inhalt einer Speicherzelle (&) oder eines Registers (R) von einem Register (*Bsp:* `sub.b #5, R15` *oder* `sub.b &40, R15`)
inc.b	Reg	Inkrementiert den Inhalt eines Registers (*Bsp:* `inc.b R15`)
cmp.b	Wert, Reg	Vergleicht ("compare") einen Wert (#) oder den Inhalt einer Speicherzelle (&) oder eines Registers (R) mit dem Inhalt eines Registers (*Bsp:* `cmp.b #5, R15` *oder* `cmp.b &40, R15`)
jmp	Label	Springt bedinungslos zu einem Label
jc / jnc	Label	Springt zu einem Label, falls das Carry-Bit (nicht) gesetzt ist
jz / jnz	Label	Springt zu einem Label, falls das Zero-Bit (nicht) gesetzt ist
jn	Label	Springt zu einem Label, falls das Negative-Bit gesetzt ist
nop		"No Operation". Es passiert nichts.
sleep		Makro: Dient dazu, die Befehlsausführung zu verlangsamen. Es passiert für etwa eine halbe Sekunde nichts.
ledXon		Makro: Schaltet die LED X (1-8) ein, ohne die anderen LEDs zu beeinflussen. (*Bsp:* `led1on`)
ledXoff		Makro: Schaltet die LED X (1-8) aus, ohne die anderen LEDs zu beeinflussen. (*Bsp:* `led3off`)
testBtnX		Makro: Überprüft, ob Taster X (1-4) momentan gedrückt ist und setzt in diesem Fall das Zero-Bit.
hold		Makro: Beendet die Ausführung des Programms.

Vorlagen und Makros

Die für die Lösung der Aufgaben benötigten Vorlagen sind als Zip-Daten exportierte Projekte des Code-Composer-Studios und sind dort auch zu importieren[11]. Da die Vorlagen mit dem Projekt wuchsen sollten mit der letzten Vorlage auch frühere Aufgaben lösbar seinen, jedoch sind nicht alle dort enthaltenen Makros zur Bearbeitung aller Aufgaben notwendig. Das folgende Listing zeigt den Quelltext der verwendeten Makros:

```
hold    .macro                                                  1
__hold?   jmp __hold?                                           2
.endm                                                           3
                                                                4
sleep   .macro                                                  5
      call  #_sleep                                             6
.endm                                                           7
                                                                8
setP1out  .macro                                                9
      bis.b   #11111111b,&P1DIR       ; define output           10
      bic.b   #11111111b,&P1OUT                                 11
.endm                                                           12
                                                                13
setP2in   .macro                                                14
      bis.b   #00000000b,&P2DIR       ; define input            15
      bis.b   #00111111b,&P2REN       ; enable pullup/          16
         pulldown resistors
      bis.b   #00001111b,&P2OUT       ; set pullup/pulldown     17
         resistors 0-3 to pullup
      bic.b   #11110000b,&P2OUT       ; set pullup/pulldown     18
         resistors 4-7 to pulldown
.endm                                                           19
                                                                20
testBtn1  .macro                                                21
      bit.b #00000001b, &P2IN                                   22
.endm                                                           23
                                                                24
                                                                25
[...]                                                           26
                                                                27
led1on    .macro                                                28
      bis.b #00000001b, &P1OUT                                  29
.endm                                                           30
                                                                31
[...]                                                           32
                                                                33
led1off   .macro                                                34
```

[11] File - Import... - Code Composer Studio - Existing CCS Eclipse Project

```
      bic.b #00000001b, &P1OUT                                          35
.endm                                                                   36
                                                                        37
[...]                                                                   38
                                                                        39
init    .macro                                                          40
      mov       #0280h,SP                 ; initialize stack          41
         pointer
      mov     #WDTPW+WDTHOLD,&WDTCTL   ; stop watchdog timer            42
      setP1out                                                          43
      setP2in                                                           44
.endm                                                                   45
                                                                        46
                                                                        47
                                                                        48
                                                                        49
.text                                                                   50
      .global _main           ; define entry point                      51
_sleep    mov #0, R14                                                   52
__sl    dec R14                                                         53
      jnz __sl                                                          54
      ret                                                               55
```

TUM12 14.12.2012

1. Klausur aus der Informatik

Für alle Aufgaben ist eine Registermaschine mit folgendem Befehlssatz anzunehmen:

LOAD x	kopiert den Wert aus Speicherzelle x in den Akkumulator
DLOAD i	lädt die Ganzzahl i in den Akkumulator
STORE x	speichert den Wert aus dem Akkumulator in Speicherzelle x
ADD x	addiert den Wert aus Speicherzelle x zum Wert im Akkumulator und legt das Ergebnis dort ab
SUB x	subtrahiert den Wert aus Speicherzelle x vom Wert im Akkumulator und legt das Ergebnis dort ab
DADD i	addiert den Wert i zum Wert im Akkumulator und legt das Ergebnis dort ab
DSUB i	subtrahiert den Wert i vom Wert im Akkumulator und legt das Ergebnis dort ab
DIV x	dividiert ganzzahlig den Wert im Akkumulator durch den Wert in Speicherzelle x und legt das Ergebnis im Akkumulator ab. (z.B. 17:5 → 3)
MOD x	dividiert den Wert im Akkumulator durch den Wert in Speicherzelle x und legt den Rest der ganzzahligen Division im Akkumulator ab. (z.B. 17:5 → 2)
JMPP x	springt zum Befehl in Speicheradresse x, falls der Wert im Akkumulator größer als 0 ist.
JMPZ x	springt zum Befehl in Speicheradresse x, falls der Wert im Akkumulator 0 ist.
JMPN x	springt zum Befehl in Speicheradresse x, falls der Wert im Akkumulator kleiner als 0 ist.
JMP x	führt einen unbedingten Sprung zum Befehl in Speicherzelle x aus
HOLD	Beendet die Ausführung des Programms

Aufgabe 1 — 3+4 Punkte

a) In modernen Prozessoren (CPU) sind Rechenwerk und Steuerwerk in einem gemeinsamen Chip verbaut, obwohl sie klar getrennte Aufgaben haben. Erläutern Sie kurz die Aufgabe des Rechenwerks und geben Sie die Befehle der Registermaschine an, bei welchen das Rechenwerk aktiv wird.

b) Die Registermaschine führt entsprechend dem Von-Neumann-Zyklus bei der Abarbeitung eines Befehls die folgenden Phasen aus: UPDATE PROGRAM COUNTER, FETCH, EXECUTE, FETCH OPERANDS, DECODE,. Bringen sie die Phasen in die korrekte Reihenfolge und erklären Sie kurz die Phasen DECODE und FETCH. Gehen Sie dabei auch auf die in den Phasen betroffenen Register ein.

Aufgabe 2 — 4+2 Punkte

Gegeben ist folgendes Programm:

```
0:  LOAD 100
1:  SUB 101
2:  JMPN 6
3:  LOAD 100
4:  STORE 102
5:  HOLD
6:  LOAD 101
7:  STORE 102
8:  HOLD
```

Der Zustand der Registermaschine ist im Folgenden vereinfachend durch die Inhalte des Akkumulators A, des Befehlszählers BZ sowie der Speicherzellen 100 bis 102 beschrieben. Dabei ist zu Beginn der Akku und der Befehlszähler mit dem Wert 0 belegt.

a) Geben sie tabellarisch den Ablauf der Zustände der Registermaschine bei einem Programmdurchlauf an, wenn die Speicherzelle 100 mit dem Wert 23, die Speicherzelle 101 mit dem Wert 5 und die Speicherzelle 102 mit dem Wert 0 vorbelegt sind. Geben Sie auch immer den Befehl mit an, der die Zustandsänderung bewirkt.

b) Geben Sie eine mögliche Vorbelegung der Speicherzelle 100 bis 102 an, damit nach Programmablauf der ursprünglich in Speicherzelle 101 vorhandene Wert auch in Speicherzelle 102 gespeichert ist.

Aufgabe 3 8 Punkte

Schreiben Sie ein Programm für die Registermaschine, welches überprüft, ob 3 Zahlen identisch sind. Die Zahlen sind in den Speicherzellen 100, 101 und 102 abgelegt. Falls die Werte gleich sind, soll das Programm in der Speicherzelle 103 den Wert 1 speichern, ansonsten den Wert 0. Geben Sie zu jedem Befehl auch dessen Speicheradresse an.

Aufgabe 4 10 Punkte

Nachfolgend ist der euklidische Algorithmus angegeben. Er berechnet den größten gemeinsamen Teiler zweier Zahlen a und b und speichert das Ergebnis in der Variablen e. Dabei berechnet die Operation „mod“ den Rest der ganzzahligen Division.

```
solange b ≠ 0
        h = a mod b
        a = b
        b = h
e = a
```

Setzen Sie den Algorithmus für die Registermaschine um. Nutzen sie dazu folgende Speicherzellen: a=100, b=101, h=102, e=103. Geben Sie zu jedem Befehl auch die Speicheradresse an.

Aufgabe 5 6 Punkte

Gegeben ist folgendes Programm einer Registermaschine:

```
1:    DLOAD 0
2:    STORE 100
3:    STORE 101
4:    LOAD 100
5:    DADD 1
6:    STORE 100
7:    LOAD 101
8:    ADD 100
9:    STORE 101
10:   DLOAD 1000
11:   SUB 100
12:   JMPP 4
13:   HOLD
```

Geben sie den Algorithmus in einer Hochsprache ihrer Wahl (z.B. Java, Python) an. Benennen Sie dazu die Variablen x für Speicherzelle 101 und i für Speicherzelle 100.

Viel Erfolg!

Literatur

[1] *MSP430^{TM}16-bit Ultra-Low Power MCUs - Value Line - MSP430G2253 - TI.com*. http://www.ti.com/product/msp430g2253. `http://www.ti.com/product/msp430g2253`

[2] Brichzin, Peter ; Freiberger, Ulrich ; Reinold, Klaus ; Wiedemann, Albert: *Informatik Oberstufe 2: Maschinenkommunikation - Theoretische Informatik*. Oldenbourg Schulbuchverlag, 2010. – ISBN 3637008300

[3] Hubwieser, Peter ; Löffler, Patrick ; Schwaiger, Petra: *Informatik - Ausgabe für Bayern und Nordrhein-Westfalen: Informatik. Schülerbuch 12. Klasse. Ausgabe für Bayern: Formale Sprachen, Kommunikation und ... Rechners, Grenzen der Berechenbarkeit: 5*. 1., Aufl. Klett, 2010. – ISBN 3127310684

[4] Schwaiger, Petra ; Voss, Siglinde ; Wagner, Andreas ; Stefan, Winter: *Handreichung Informatik am Naturwissenschaftlich-technologischen Gymnasium, Jahrgangsstufe 12*. `http://www.isb.bayern.de/schulartspezifisches/materialien/informatik-naturwissenschaftlich-jgst-12/`

Lizenz