Hardwarearchitekturen und Rechensysteme

Sommersemester 2007

Kurzübersicht MC68000 Mikroprozessor

Dokumentversion 1.0

Sebastian Steinhorst

Dieses Dokument erhebt keinen Anspruch auf Vollständigkeit, sondern soll als kompakte Referenz für einen Einstieg in die Assemblerprogrammierung mit dem MC68000-Prozessor dienen.

1 Struktur des Operationswerks

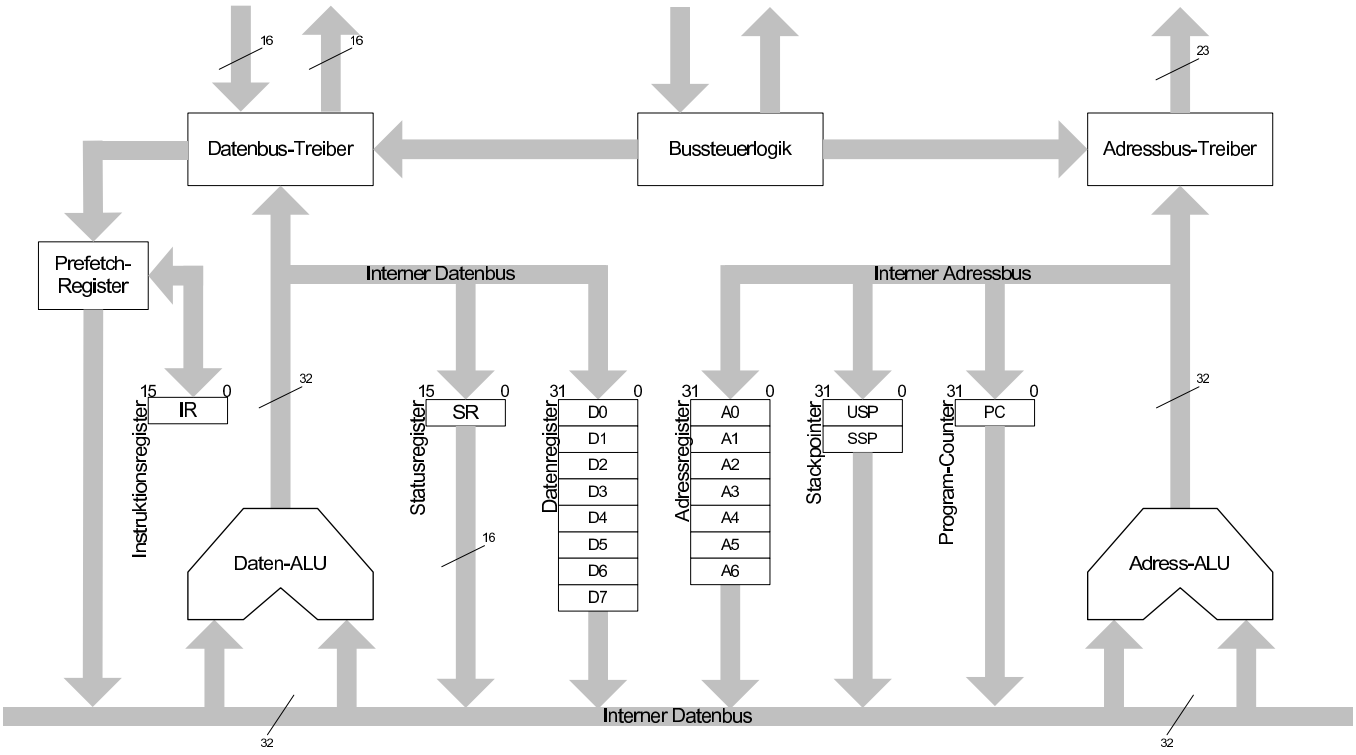


Abbildung 1: Blockschaltbild des Operationswerks des MC68000 Mikroprozessors

2 Registermodell/Programmiermodell

Als Programmiermodell werden die Struktur und die Funktion der Register bezeichnet, die von einem Programm direkt angesprochen werden können. Der MC68000 besitzt aus Sicht der Programmierung acht 32-Bit Datenregister D0 - D7. Es gibt sieben 32-Bit breite Adressregister A0 - A6 und jeweils einen 32-Bit User- und einen Supervisor-Stackzeiger USP bzw. SSP im Adressregister A7 und A7'. Darüber hinaus gibt es den 32-Bit Programmzähler PC und das 16-Bit Statusregister SR, dessen niederwertige 8 Bit das User-Byte und die höherwertigen 8 Bit das System-Byte darstellen. Abbildung 2 stellt das Programmiermodell grafisch dar.

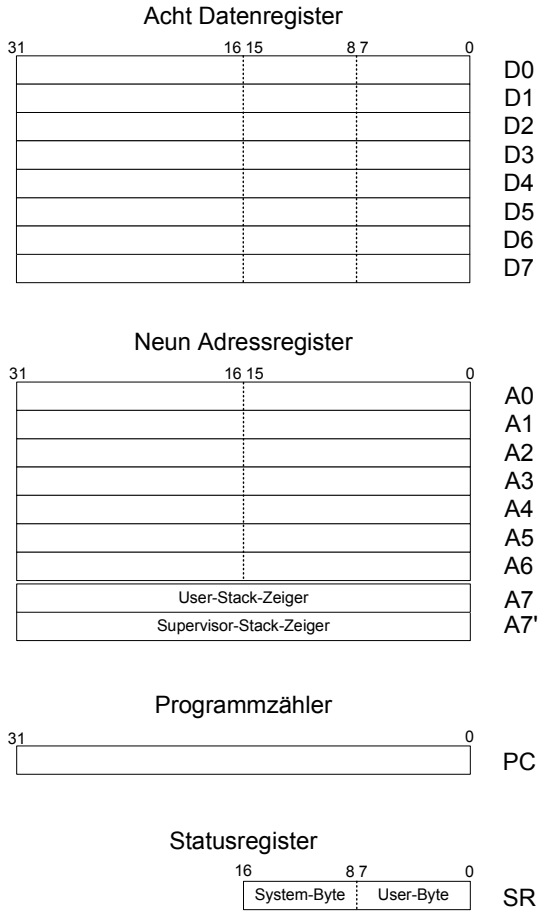


Abbildung 2: Programmiermodell des MC68000 Mikroprozessors

2.1 Aufbau des Statusregisters

Das Statusregister beinhaltet die Status-Flags des Prozessors. Aufgeteilt ist es in ein System-Byte und ein User-Byte. Das User-Byte besitzt mit seinen niederwertigsten 5 Bit die Flag-Bits für die Bedingungscodes. Verschiedene Operationen setzen diese Flags und ermöglichen im MC68000-Prozessor so von ihrer Belegung abhängige, also bedingte, Operationen. Abbildung 3 zeigt den Aufbau des Statusregisters.

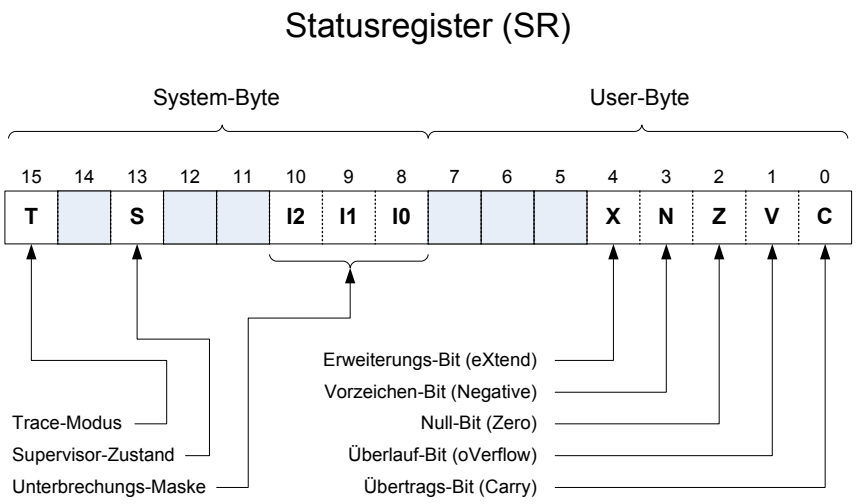


Abbildung 3: Statusregister des MC68000 Mikroprozessors

3 Speichermodell

Der Speicher ist für den 68000-Prozessor in 8-Bit-Einheiten unterteilt, die pro Byte durchnummeriert sind. Eine Speichereinheit aus zwei Byte wird als Wort bezeichnet und besitzt dann bei einer Größe von 16 Bit ein höherwertiges und ein niederwertiges Byte. Zwei Wörter bilden ein Doppelwort oder Langwort mit 32 Bit. Im Folgenden werden die wesentlichen Adressierungsarten des Prozessors betrachtet. Neben den nachfolgend erläuterten gibt es noch komplexere weitere Adressierungsarten, die in den zu entwickelnden Programmen im Rahmen der Vorlesung HWR keine Anwendung finden werden.

3.1 Unmittelbare Adressierung

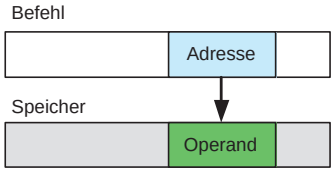
Bei der unmittelbaren Adressierung wird der Operand dem Befehl als Konstante übergeben.

Beispiel: `MOVE.W #$FFFF,D0` (Schreibe den hexadezimalen Wert FFFF in das Register D0.)

Damit ein übergebener Zahlwert als konstanter Operand interpretiert wird, muss das Zeichen `#` vorangestellt werden. Das Zeichen `$` führt dazu, dass der folgende Wert als hexadezimal interpretiert wird. Andernfalls wird er dezimal interpretiert.

3.2 Absolute Adressierung

Bei der absoluten Adressierung wird die effektive Speicheradresse direkt als Operand einer Operation angegeben.

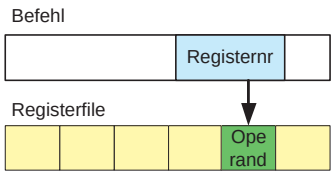


Beispiel: `MOVE.W $1000,$2000` (Schreibe Inhalt des Datenworts von Speicheradresse $1000 nach Adresse $2000.)

Wie man erkennt, bezieht sich die Angabe eines Zahlenwertes ohne vorangehendes `#` auf eine Speicheradresse. Das Zeichen `$` führt zu einer hexadezimalen Interpretation der angegebenen Adresse.

3.3 Register-direkte Adressierung

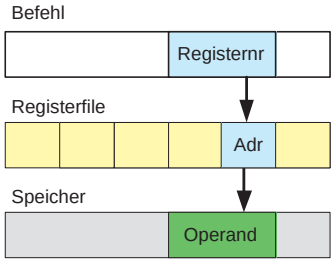
Bei der Register-direkten Adressierung wird der Operand in einem Register übergeben.



Beispiel: `MOVE.W D0,D1` (Schreibe den Inhalt des Registers D0 in das Register D1.)

3.4 Register-indirekte Adressierung

Bei der Register-indirekten Adressierung zeigt der Inhalt eines Adressregisters auf die Speicheradresse eines Operanden.



Beispiel: `MOVE.W (A0),D0` (Schreibe das Datenwort, das an der im Adressregister A0 angegebenen Speicheradresse steht, in das Register D0.)

3.4.1 Postinkrement bzw. -dekrement

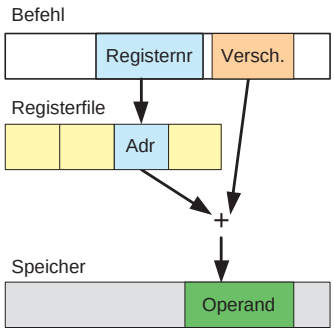
Bei der Register-indirekten Adressierung mit Postinkrement bzw. -dekrement wird der Inhalt des Adressregisters nach dem Ausführen der Operation um die Größe des Datenelements erhöht bzw. erniedrigt.
Beispiel: `MOVE.W (A0)+,D0` (Schreibe das Datenwort, das an der im Adressregister A0 angegebenen Speicheradresse steht, in das Register D0 und erhöhe den Wert von A0 um eine Datenwortadresse, also um 2.)

3.4.2 Preinkrement bzw. -dekrement

Bei der Register-indirekten Adressierung mit Preinkrement bzw. -dekrement wird der Inhalt des Adressregisters vor dem Ausführen der Operation um die Größe des Datenelements erhöht bzw. erniedrigt.
Beispiel: `MOVE.W +(A0),D0` (Erhöhe den Wert von A0 um eine Datenwortadresse, also um 2 und schreibe dann das Datenwort, das an der im Adressregister A0 nun angegebenen Speicheradresse steht, in das Register D0.)

3.4.3 Register-indirekte Adressierung mit Adressdistanz (Displacement)

Bei der Register-indirekten Adressierung mit Displacement zeigt der Inhalt eines Adressregisters, verschoben um einen angegebenen Wert, auf die Speicheradresse eines Operanden.



Beispiel: `MOVE.W 300(A0),D0` (Schreibe das Datenwort an der im Adressregister A0 angegebenen Speicheradresse, erhöht um den Wert 300, in das Register D0.)

3.5 weitere Adressierungsarten

- Register-indirekte Adressierung mit Index und Adressdistanz
- Programmzähler-bezogene Adressierung mit Adressdistanz
- Programmzähler-bezogene Adressierung mit Index und Adressdistanz

4 Programmierung mit Assembler

Der eigentliche vom Prozessor verarbeitete Programmcode ist ein binärer Maschinencode, der in einer Prozessor-spezifischen Codierung vorliegt. Die Programmierung in Assembler erlaubt es, mit mnemonischen, also verständliche Assoziationen erlaubenden, Bezeichnungen für die einzelnen Operationen und Operanden eine Programmierung in Textform vorzunehmen. Dieser mnemonische Code wird beim Assemblieren wieder in Maschinencode übersetzt, wobei die Programmierung erleichternde Zusätze wie z.B. Sprungmarken durch Einsetzung von effektiven Adressen aufgelöst werden.

4.1 Befehlssatz des MC68000

Der Befehlssatz des MC68000 gliedert sich in verschiedene Operationsgruppen:

- Datenbewegung
- Integer-Arithmetik
- Logische Operationen
- Schiebe- und Rotationsoperationen
- Bittest und Bitmanipulation
- Operationen für BCD-Zahlen
- Kontrolle des Programmflusses
- Systemsteuerbefehle

Für eine vollständige detaillierte Auflistung der Operationen des MC68000-Prozessors wird auf eine der Literaturquellen verwiesen. Eine sehr gute Quickreference der Operationen des MC68000 findet sich unter [Kel04].

4.2 Programmierkonzept

Im Folgenden sollen aus praktischer Sicht die Grundlagen zum Schreiben, Assemblieren und Ausführen eines MC68000-Programms vermittelt werden. Der MC68000 besitzt einen sogenannten 2-Adress-Code, sodass bei Ergebnis-liefernden Operationen mit zwei Operanden das Ergebnis in den zweiten Operanden, den Zieloperanden, geschrieben wird. Für viele Operationen kann durch das Anhängen der Zusätze `.B`, `.W` oder `.L` entschieden werden, ob sie sich auf ein Byte (8-Bit), Wort (16-Bit) oder Langwort (32-Bit) beziehen, wobei die Operandengröße vom niederwertigsten Bit aus gezählt wird. Die Standard-Datengröße ist das 16-Bit-Wort, deshalb kann auf die Angabe des Zusatzes `.W` auch verzichtet werden. Manche Operationen, wie beispielweise `DIVU`, besitzen eine feste Datengröße der Operanden.

4.2.1 Grundgerüst eines MC68000-Assemblerprogramms

Ein Assemblerprogramm besitzt vier durch Tabs getrennte Spalten:

1. Marken
2. Operation
3. Operanden (getrennt durch Kommas)
4. Kommentar

Jedes Programm sollte zu Beginn die folgenden vier Operationen enthalten, um den Stackzeiger zu initialisieren, um die Programmstartadresse im Speicher zu definieren und so auch bei einem Reset wieder neu beginnen zu können:

```
        ORG     $0              Setze absolute Adresse im Speicher auf 0
        DC.L    $8000           Wert des Stackzeigers nach einem Reset
        DC.L    START           Wert des Programmzählers nach einem Reset
        ORG     $2000           Beginne mit dem Programmcode im Speicher an
                                Adresse 2000 Hex

START   ...     ...             Startmarke für den Beginn des eigentlichen
                                Programms
        BREAK                   Programmabarbeitung unterbrechen (sonst läuft
                                der Programmzähler über diese Stelle weiter)
```

4.2.2 Datenzuweisungen und -bewegungen

```
        MOVE    #$1234,D0       Schreibe hexadezimalen Wert 1234 als Wort
                                (16 Bit) in Register D0. Die allgemeine
                                MOVE-Operation setzt die Statusregistereinträge
                                N und Z entsprechend dem Quelloperanden und
                                setzt V und C auf 0.

        MOVE    $1234,D0        Schreibe 16-Bit Datenwort an Speicheradresse
                                $1234 nach Register D1

        MOVE.L  D0,D1           Schreibe 32-Bit Langwort aus Register D0
                                nach Register D1

        EXG.L   D0,D1           Werte von D0 und D1 austauschen

        CLR.L   D0              Initialisiert Register D0 mit Wert 0

        CLR     D0              Setzt das niederwertige Wort in D0 auf 0

        SWAP    D0              Vertauschung von Registerhälften. Das niederwertige
                                und das höherwertige Teilwort eines Datenregisters
                                werden miteinander vertauscht.

        MOVEM   D0-D7,-(A7)     Sequentielles Schreiben der Register D0 - D7 in den
                                mit A7 adressierten Speicherbereich (Stack). Dies
                                dient zur Sicherung des Inhaltes z. B. bei Unter-
                                programmverzweigungen.

        MOVEM   +(A7),D0-D7     Sequentielles Restaurieren der Register D0 - D7 aus
                                dem mit A7 adressierten Speicherbereich (Stack).
                                Dies dient zur Wiederherstellung des Inhaltes z. B.
                                bei Unterprogrammverzweigungen.

        MOVEA.L A6,A7           Adress-Kopieren von A6 nach A7 ohne Modifikation
                                des Statusregisters

        LEA  MYVAR,A1           Laden einer effektiven 32-Bit Speicheradresse in ein
```

```
                                    Adressregister (Load Effective Address). Hier die
                                    Adresse der Variablenspeicherplatzes MYVAR in
                                    Adressregister A1.
```

4.2.3 Konstanten und Variablen in Speicherbereichen

Die nachfolgenden Befehle `DC` und `DS` zum Speichern von Konstanten und Reservieren von Speicherbereichen als Variablenspeicher belegen den Speicher direkt an der Position, an der sie sich im Programmablauf des Programmspeichers befinden. Somit sollten sie vor dem Start des eigentlichen Programmcode oder danach definiert werden, da sonst der Inhalt dieser Speicherbereiche als Programmcode interpretiert würde.

```
MYCONST DC      $FFFF           Mit DC (Define Constant) wird eine Konstante
                                im Speicher abgelegt (hier hexadezimal FFFF).
                                Sie kann mit dem nutzerdefinierten Label MYCONST
                                an jeder Stelle im Programm eingesetzt werden.
                                Beispiel: MOVE MYCONST, D0

MYVAR   DS.L    1               Mit DS (Define Space) wird ein Speicherbereich
                                reserviert, hier wegen des Parameters 1 genau
                                ein Doppelwort. Dieser Speicherbereich kann
                                an jeder Stelle des Programms mit dem nutzer-
                                definierten Label MYVAR verwendet werden.
                                Beispiel: MOVE.L  $1111, MYVAR
                                          ADD.L   #1,MYVAR
```

4.2.4 Ganzahl-Berechnungen

```
        ADD.L   #2,D0           Addiere auf den Wert des Registers D0 den
                                Wert 2 (D0 = D0 + 2)

        SUB.L   #1,D0           Subtrahiere vom Wert des Registers D0 den
                                Wert 1 (D0 = D0 - 1)

        MULU    #3,D0           Vorzeichenloses Multiplizieren des Registers
                                D0 mit dem Wert 3 (D0 = D0 * 3). Die Operanden
                                sind 16 Bit, das Ergebnis belegt 32 Bit.

        MULS    #3,D0           Vorzeichenbehaftetes Multiplizieren des Registers
                                D0 mit dem Wert 3 (D0 = D0 * 3). Die Operanden
                                sind 16 Bit, das Ergebnis belegt 32 Bit.

        DIVU    #4,D0           Vorzeichenloses Dividieren des Registers
                                D0 durch den Wert 4 (D0 = D0 / 4)
                                Der 16-Bit Zieloperand (hier D0) wird durch
                                den 16-Bit Quelloperanden geteilt. Die
```

niederwertigen 16 Bit des Zielregisters erhalten den Quotienten, die höherwertigen 16 Bit erhalten den Rest der Division.

```
DIVS    #4,D0
```
Vorzeichenbehaftetes Dividieren des Registers D0 durch den Wert 4 (D0 = D0 / 4)
Der 16-Bit Zieloperand (hier D0) wird durch den 16-Bit Quelloperanden geteilt. Die niederwertigen 16 Bit des Zielregisters erhalten den Quotienten, die höherwertigen 16 Bit erhalten den Rest der Division. Das Vorzeichen des Restes entspricht dem des Dividenden.

4.2.5 Logische Operationen

```
AND.L   D0,D1
```
Logisches UND von D0 und D1 (D1 = D0 AND D1)

```
OR.L    D0,D1
```
Logisches ODER von D0 und D1 (D1 = D0 OR D1)

```
NOT.L   D0
```
Logische Negation (Einerkomplement) von D0 (D0 = NOT D0)

```
EOR.L   D0,D1
```
Logisches EXKLUSIV-ODER von D0 und D1 (D1 = D0 EOR D1)

4.2.6 Schiebe- und Rotierbefehle

```
ASL.L   #3,D0
```
Arithmetisches bitweises Linksschieben (hier um 3 Stellen). Von rechts wird mit 0 aufgefüllt und höchstwertiges Bit wird beim Verschieben in Statusregister C und X geschrieben.

```
ASR.L   #3,D0
```
Arithmetisches bitweises Rechtsschieben (hier um 3 Stellen). Von links wird mit dem höchstwertigen Bit aufgefüllt und niedrigstwertiges Bit wird beim Verschieben in Statusregister C und X geschrieben.
Das Vorzeichenbit wird somit erhalten.

```
LSL.L   #3,D0
```
Logisches bitweises Linksschieben (hier um 3 Stellen). Von rechts wird mit 0 aufgefüllt und höchstwertiges Bit wird beim Verschieben in Statusregister C und X geschrieben.

```
LSR.L   #3,D0
```
Logisches bitweises Rechtsschieben (hier um 3 Stellen). Von links wird mit 0 aufgefüllt und das niedrigstwertige Bit wird beim Verschieben in Statusregister C und X geschrieben.

```
ROL.L   #3,D0          Linksrotation (hier um 3 Stellen). Es erfolgt
                       eine Ringverschiebung, so dass jeweils alle
                       Bitstellen nach links verschoben werden und die
                       aus dem Register geschobenen höchstwertigen Bits
                       das Register von rechts auffüllen. Das höchst-
                       wertige verschobene Bit schreibt jeweils C im
                       Statusregister.

ROR.L   #3,D0          Rechtsrotation (hier um 3 Stellen). Es erfolgt
                       eine Ringverschiebung, so dass jeweils alle
                       Bitstellen nach rechts verschoben werden und
                       die aus dem Register geschobenen niedrigstwertigen
                       Bits das Register von links auffüllen. Das
                       niedrigstwertige verschobene Bit schreibt jeweils
                       C im Statusregister.
```

4.2.7 Verzweigungen

```
CMP.L   D0,D1          Vergleich: D1-D0 und setzen der Statusregister-
                       Flags N (negative), Z (zero), V (overflow) und
                       C (carry) entsprechend dem Subtraktionsergebnis.

BGE     START          Springe zur angegebenen Marke, wenn Auswertung
                       der Z- und V- Flags die ">="-Bedingung erfüllt:
                       N & V + !N & !V (branch if greater or equal)
```

Zusammenfassung der wesentlichen bedingten Verzweigungen abhängig von der Auswertung des Statusregisters:

Operation	Erklärung	Auswertung des SR
BNE	branch if not equal	$!Z$
BEQ	branch if equal	Z
BPL	branch if plus	$!N$
BMI	branch if minus	N
BLT	branch if less than	$N \,\&\, !V + !N \,\&\, V$
BGT	branch if greater than	$N \,\&\, V \,\&\, !Z + !N \,\&\, !V \,\&\, !Z$
BLE	branch if less or equal	$Z + N \,\&\, !V + !N \,\&\, V$
BGE	branch if greater or equal	$N \,\&\, V + !N \,\&\, !V$

4.2.8 Subroutinen mit Parameterübergabe per Stack

Mittels Subroutinen kann eine Strukturierung und prozedurale Wiederverwendung von Befehlssequenzen erzeugt werden. Mit dem Befehl `BSR` wird Programmzähler-relativ zu einer Sprungmarke verzweigt und der dortige Code ausgeführt, bis der Befehl `RTS` den Rücksprung in die aufrufende Programmsequenz herbeiführt. Zu beachten ist, dass eine Subroutine den Inhalt der Register nach ihrer Beendigung so zurücklassen sollte, wie sie ihn bei ihrem Beginn vorgefunden hat. Eine eventuelle Parameterübergabe in Registern ist davon ausgenommen.

Der folgende Assemblercode zeigt eine Unterprogramm-Verzweigung mit einer Parameterübergabe "by value" über den Stack. Auf den Code folgt eine Erläuterung im Text.

```
1  HAUPTPR ...
2          MOVE.L  #$12,-(A7)      Lege Parameter 1 auf den Stack
3          MOVE.L  #$23,-(A7)      Lege Parameter 2 auf den Stack
4          MOVE.L  #$34,-(A7)      Lege Parameter 3 auf den Stack
5          MOVE.L  #$45,-(A7)      Lege Parameter 4 auf den Stack
6          MOVEA.L A7,A6           Sichern des Stackzeigers in A6
7          BSR     UNTERPR         Springe zur Subroutine UNTERPR
8                                  (branch to subroutine)
9          MOVEA.L A6,A7           Restaurieren des Stackzeigers aus A6
10         ...
11         BREAK                   Ausführung unterbrechen
12
13 UNTERPR MOVEM   D0-D7,-(A7)     Retten der Register D0 - D7 auf Stack
14         MOVE.L  (A6)+,D3        Hole Parameter 4 vom Stack in D3
15         MOVE.L  (A6)+,D2        Hole Parameter 3 vom Stack in D2
16         MOVE.L  (A6)+,D1        Hole Parameter 2 vom Stack in D1
17         MOVE.L  (A6)+,D0        Hole Parameter 1 vom Stack in D0
18
19         ...                     Eigentlicher Code des Unterprogramms
20
21         MOVEM   (A7)+,D0-D7     Restaurieren der Register D0 - D7
22         RTS                     Springe zurück (return from
23                                 subroutine)
```

Zur Übergabe der Parameter werden diese in den Zeilen 2 bis 5 auf den Stack gelegt, indem der Befehl `MOVE` Adressregister-indirekt mit Predekrementierung jeweils den Stack-Zeiger vor dem Schreiben des Parameters dekrementiert. In Zeile 6 wird der Stack-Zeiger gesichert. In Zeile 7 wird nun zum Unterprogramm verzweigt. Dort werden mit dem speziellen Befehl `MOVEM` in Zeile 13 die Register D0 bis D7 auf den Stack gesichert. Nun werden Adressregister-indirekt mit Postinkrementierung Parameter1 bis Parameter4 vom Stack geholt, indem mit A6 adressiert wird und in die Register D0 bis D3 geschrieben (Zeilen 14-17). Zu beachten ist die umgekehrte Reihenfolge zum Legen auf den Stack wegen der LIFO(Last-In-First-Out)-Eigenschaft des Stacks.

Nun kann die eigentliche Befehlssequenz des Unterprogramms mit den Parametern abgearbeitet werden (angedeutet durch Punkte in Zeile 19). Am Ende des Unterprogramms wird in Zeile 21 mit `MOVEM` nun wieder Adressregister-indirekt mit Postinkrementierung der Inhalt der Register D0 bis D7 restauriert und mit `RTS` in Zeile 22 der Rücksprung zum Hauptprogramm durchgeführt. Dort wird in Zeile 9 nun der User-Stack-Pointer wieder aus A6 restauriert. Damit sind alle Register sowie der Stack-Zeiger wieder im selben Zustand wie vor der Vorbereitung des Verzweigens ins Unterprogramm.

4.3 Übersetzung und Ausführung von Programmen mit dem MC68000-Simulator BSVC

Mit BSVC existiert ein freier Simulator für den MC68000, den es sowohl für Linux als auch Windows gibt. Die Homepage von BSVC befindet sich unter `www4.ncsu.edu/~bwmott/www/bsvc`. Da

dort die Linux-Version nicht als Binärpaket vorliegt und erst kompiliert werden muss, findet sich im Downloadbereich der Vorlesung auf der Entwurfsmethodik-Website eine bereits für aktuelle x86-Linux-Systeme kompilierte Version des BSVC-Simulators. Es empfiehlt sich die Verwendung des Install-Skripts `install.sh`, da BSVC im Verzeichnis `~/bsvc` installiert werden muss.

Die BSVC-Installation bringt einen Crossassembler mit, der das in einer Textdatei vorliegende Programm in den Maschinencode des Prozessors übersetzt. Ein z.B. in der Textdatei *myprog.s* erstelltes Assemblerprogramm wird mit dem Kommando `68kasm -l myprog.s` assembliert.

Nun kann es in den Simulator geladen werden, der unter Linux im Programmverzeichnis mit `./bsvc` gestartet wird. Unter *File* muss zunächst mit *Open Setup...* eine Konfiguration für den Simulator geladen werden. Dazu wird im Unterverzeichnis *samples/m68000* die Datei *simple.setup* geöffnet. Danach kann mit *Load Program* aus dem Menüpunkt *File* das assemblierte Programm geladen werden, das die Dateinamenserweiterung *.h68* besitzt. Die Schaltfläche *Reset* initialisiert das Programm. Mit der Schaltfläche *Run* kann das Programm nun gestartet werden. Mit der Schaltfläche *Single Step* kann auch schrittweise durch das Programm gelaufen werden, um die Operationen einzeln verfolgen zu können.

Für Windows existiert mit EASy68K ein sehr benutzerfreundlicher Assembler und Simulator für den MC68000. Auf der Webseite des Projekts unter [Kel] findet sich ein Installationspaket. Das Programm besitzt eine durchgehend grafische Oberfläche und eine Online-Hilfe mit Einführung.

Literatur

[faq] FAQ zum Motorola 68000. URL: `http://www.ee.ualberta.ca/archive/m68kfaq.html`.

[Kan85] G. Kane. *68000 Mikroprozessorhandbuch.* McGraw-Hill, Hamburg, 1985.

[Kel] C. Kelly. EASy68K Assembler und Simulator. URL: `http://www.monroeccc.edu/ckelly/EASy68K.htm`.

[Kel04] C. Kelly. Quickreference des 68000 Befehlssatzes. 2004. URL: `http://www.monroeccc.edu/ckelly/Files/EASy68KQuickRef.pdf`.

[Koc82] J. Koch. *Der 16bit-Mikroprozessor SC 68000.* Boysen + Maasch, Hamburg, 1982.

[Mot92] Motorola. Programmer's Reference Manual zum MC68000. 1992. URL: `http://www.freescale.com/files/archives/doc/ref_manual/M68000PRM.pdf`.

[Neu88] J. Neuschwander. *Struktur und Programmierung eines Mikroprozessorsystems.* R. Oldenbourg Verlag, München, 1988.

[Sca83] L. Scanlon. *Die 68'000er - Grundlagen der Programmierung.* AT Verlag, Aarau, Schweiz, 1983.

[Vie85] C. Vieillefond. *Programmierung des 68000.* Sybex-Verlag GmbH, Düsseldorf, 1985.