DIE ASSEMBLER-ENTWICKLUNGSUMGEBUNG

1 Hardware

Als Zielrechner wird ein Spezialrechner verwendet, den wir als „*mx*“ bezeichnen – eine Anspielung an *MinMax*, das Übungs-Betriebssystem für die Praktika zu Informatik 4. Die Zielrechner besitzen einen Motorola MC68000 als CPU und sind für Ein- und Ausgabe über eine serielle Schnittstelle mit einer *Sun Ray* verbunden. Gestartet wird ein Programm auf dem Zielrechner ebenfalls über die serielle Schnittstelle. Jede Sun Ray im EN268 hat ihren eigenen *mx*. Bild 1 zeigt diesen Aufbau.

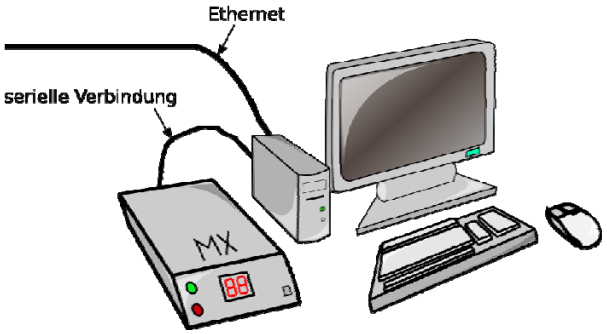

Bild 1: Die Assembler-Entwicklungsumgebung

Die Zielrechner befinden sich alle im EN268. Zum Übersetzen kann eine beliebige an das Fachbereichsnetz angeschlossene Station benutzt werden, gestartet werden können die Programme dagegen nur von der Workstation, an der der *mx* angeschlossen ist.

2 Die Programmentwicklung

Bei der Programmentwicklung in Assembler wird der GNU-Assembler **`as`** verwendet, der für die Assembler-Übung als Cross-Assembler übersetzt wurde. Er kann MC68000-Assembler-Programme für den *mx* auf den Suns im Fachbereichsnetz übersetzen. Aufgerufen wird der Assembler aber am besten über den C-Compiler **`gcc`**, der Assembler-Quellen an ihrer Endung erkennt und den Assembler richtig aufruft. Die GNU-Programme für die *mx*-Entwicklung tragen das Präfix „**`mx-`**“, um Namensgleichheiten mit den normalen Programmen für die Suns zu vermeiden. Dieses Kapitel beschreibt die einzelnen Arbeitsschritte, die zum Schreiben und Starten eines Assemblerprogramms nötig sind.

2.1 Quelldateien

Eine Assembler-Quelldatei hat die Endung „**`.S`**“. In den Assembler-Quelldateien können Kommentare auf zwei Arten untergebracht werden:

- Alles zwischen „**`/*`**“ und „**`*/`**“ ist ein Kommentar. Diese Kommentare können nicht geschachtelt werden!
- Alles von „|“ bis zum Ende der Zeile ist ein Kommentar.

Neben den gewohnten 68000er-Assembler-Anweisungen werden noch zwei Direktiven an den Assembler benötigt:

- **`.EVEN`** weist den Assembler an, die nächste Instruktion an einer geraden Adresse abzulegen. Dies sollte vor *jeder* Prozedur stehen.
- **`.GLOBAL <Bezeichner>`** sorgt dafür, daß die Marke **`<Bezeichner>`** zum Binden exportiert wird. Nur dann kann eine Assembler-Prozedur von außen, z. B. von einer C-Funktion aus, aufgerufen werden!

C-Bezeichner erhalten vom C-Compiler immer ein „_“ vorangestellt, wenn sie als Symbole in die Objektdatei eingetragen werden. Beim Assembler muß dieser Unterstrich manuell ergänzt werden!

Daher sollte eine Assembler-Prozedur **`Beispiel`**, die aus C aufgerufen werden soll, folgendermaßen beginnen:

```
            .EVEN
            .GLOBAL _Beispiel
_Beispiel:  | hier geht es los!
```

Weitere Einzelheiten über den GNU-Assembler lassen sich in der Texinfo-Dokumentation mittels **`info as`** oder im Info-Modus des Editors **`xemacs`** nachlesen.

2.2 Kellerverwaltung

Im Keller stehen nach dem Aufruf einer Prozedur die Prozedurparameter und die Rücksprungadresse. Der SP zeigt auf das letzte gesicherte Datum, also auf die Rücksprungadresse. Der Keller wird von oben nach unten aufgebaut, wobei die Prozedurparameter nach C-Konvention von hinten nach vorne auf dem Keller abgelegt werden. Daher stehen die Parameter tatsächlich von links nach rechts gelesen an aufsteigenden Adressen im Keller. An der niedrigsten Adresse im Keller steht die Rücksprungadresse, an der höchsten der letzte Parameter.

Die erste Anweisung in der Assembler-Prozedur sollte das Retten der in der Prozedur benutzten Register sein. Sind das zum Beispiel die Datenregister 1 und 2 sowie die Adreßregister 0 bis 2, sieht das so aus:

```
MOVEM.L D1-D2/A0-A2, -(SP)
```

Es wird also jeweils der Kellerzeiger dekrementiert und das nächste Register gerettet.

Als nächstes können dann die Prozedurparameter in die Register geladen werden. Dafür benötigt man die Distanz vom aktuellen Stand des SPs bis zur Adresse des Parameters. Das kann dann z.B. so aussehen:

```
MOVE.L distanz(SP), D0
```

In C belegt *jeder* Parameter stets 4 Bytes auf dem Keller, selbst Zeichen (Typ **`char`**). Beim MC68000 sind Langworte in der Reihenfolge absteigender Byte-Wertigkeiten abgelegt, so daß ein einzelnes Zeichen im vierten Byte des Parameters auf dem Keller stehen würde:

```
MOVE.B distanz+3(SP), D0
```

Zur Vermeidung von Fehlern ist es aber zu empfehlen, alle Parameter stets wie oben als Langworte zu behandeln.

Es gibt in C keine Referenzparameter. Statt dessen übergibt man einen Zeiger auf die Resultat-Variable. Der Zeiger selbst ist dann ebenfalls 4 Bytes groß und stellt selbst einen Wertparameter dar. Der Rückgabewert einer Assembler-Funktion muß beim Rücksprung im Register **`D0`** stehen, damit C ihn wie den Rückgabewert einer C-Funktion verwenden kann. Dann darf **`D0`** natürlich nicht am Anfang auf dem Keller gesichert worden sein!

Vor dem Rücksprung von Assembler zu C werden die Registerinhalte wiederhergestellt. Das Entfernen der Parameter aus dem Keller muß nach C-Konvention der *Aufrufer* erledigen.

Angenommen, es soll die Funktion **`int abs(int x)`** von Assembler aus aufgerufen werden, die einen Parameter vom Typ **`int`** erhält und einen Rückgabewert (den Betrag von **`x`**) desselben Typs liefert. Dieser Aufruf könnte so aussehen:

```
MOVE.L  x, -(SP)        | x auf dem Keller ablegen
JSR     _abs            | Aufruf der Funktion abs()
ADDQ.L  #4,SP           | x wieder vom Keller nehmen
MOVE.L  D0,resultat     | Ergebnis ablegen
```


2.3 Übersetzen, Binden und Starten eines Programms

2.3.1 Übersetzen

Um die Datei **`Beispiel.S`** in die Objektdatei **`Beispiel.o`** zu übersetzen, erfolgt der Assembler-Aufruf mit

```
mx-gcc -m68000 -o Beispiel.o Beispiel.S
```

wie ein Aufruf des C-Compilers.

Durch den Aufruf über **`gcc`** ist es möglich, Direktiven des C-Präprozessors zu verwenden. Man kann also mit **`#define`** Makros definieren und mit **`#include`** andere Quelldateien einfügen.

2.3.2 Binden

Damit ein Assembler-Programm auf dem *mx* laufen kann, wird es mit einer Anzahl von Modulen aus MinMax gebunden und mit Modulen, die das Assembler-Programme mit Daten versorgen und seine Ergebnisse ausgeben. Diese Umgebung ist in C implementiert. Die Details dieser Implementierung spielen aber – ebenso wie die Kenntnis von C – für die Programmierung der Assembler-Aufgaben keine Rolle.

Der Binder-Aufruf geschieht über **`gcc`**. Um beispielsweise die Dateien **`Beispiel.o`** und **`Haupt.o`** zur ausführbaren Datei **`Haupt`** zu binden, würde man

```
mx-gcc -o Haupt Haupt.o Beispiel.o
```

eingeben.

2.3.3 make und das Makefile

Um die Übersetzung der Assembler-Programme zu einfach wie möglich zu gestalten, wurde ein Makefile erstellt, das alle nötigen Vorgänge automatisiert. Diese Datei heißt auch **`Makefile`** und steht, zusammen mit den Vorgaben, im Verzeichnis **`~info4/info4/aufgabe1a`** bzw. **`~info4/info4/aufgabe1c`**.

Achtung: Das Assembler-Makefile benötigt zwingend das GNU-make mit einer bestimmten Zeichenkodierung, damit Tabulatorzeichen im Makefile nicht falsch interpretiert werden. Hierzu wurde der Alias **`mxmake`** definiert, den ihr bitte verwendet!

Ein Makefile ist eine Kommandodatei für das Werkzeug **`make`** (bzw. **`mxmake`**). **`make`** dient dazu, eine oder mehrere Zieldateien auf angegebene Art und Weise aus ein oder mehreren Ursprungsdateien zu erzeugen, mitunter über eine Reihe von Zwischendateien. Ein typischer Fall ist die Übersetzung mehrerer Quelldateien (Ursprung) in eine Anzahl von Objektdateien (Zwischendateien) und schließlich das Binden in ein ausführbares Programm (Zieldatei). Das Makefile muß angeben, wie diese Dateien voneinander abhängen und mit welchen Kommandos die einzelnen Übersetzungsschritte durchgeführt werden. Als Voreinstellung

geht **make** davon aus, daß sich eine Datei mit dem Namen **Makefile** im aktuellen Verzeichnis befindet.

Wenn man das Assembler-Makefile ins aktuelle Verzeichnis kopiert hat, genügt ein simples

```
mxmake all
```

zum Übersetzen und Binden der Aufgabe in eine direkt auf dem *mx* ausführbare Datei. **mxmake** gibt dabei die Kommandos aus, die es ausführt.

2.3.4 Starten auf dem *mx*

Um das erzeugte Programm zu starten, wird das Programm **mxBoot** mit dem Namen der ausführbaren Datei (meistens **Assembler**) als erstes Argument aufgerufen:

```
mxBoot Assembler
```

mxBoot öffnet ein Fenster, das mit der seriellen Schnittstelle des *mx* an der Entwicklungs-Workstation verbunden ist, um darüber die Interaktion mit dem Programm zu ermöglichen. Danach wird die ausführbare Datei über die serielle Schnittstelle in den *mx* geladen. Sollte der Boot-Vorgang nicht automatisch starten, weil z. B. ein abgestürztes Programm den *mx* blockiert, muß der rote RESET-Taster am *mx* betätigt werden.

Auch das Starten kann mit **mxmake** automatisch erfolgen: Mittels

```
mxmake boot
```

wird das Programm, wenn erforderlich, neu übersetzt und dann mit **mxBoot** auf dem *mx* gestartet.

3 Halt! Vor dem ersten Versuch...

Zum Übersetzen und Ausführen der Programme sind noch ein paar Voreinstellungen nötig. Außerdem werden, wie schon oben erwähnt, einige Dateien vorgegeben.

3.1 Setzen der Umgebungsvariablen

Alle Programme, die für die Entwicklung auf dem *mx* zur Verfügung gestellt werden, können über das Verzeichnis **~info4/mx/bin** aufgerufen werden. Dieses Verzeichnis sollte in die Umgebungsvariable **PATH** aufgenommen werden.

Am einfachsten ist es, wenn man am Ende der Datei **.cshrc** bzw. **.bashrc** (je nach verwendeter Shell) die Zeile

```
source /home/pdv/info4/mx/lib/mxenv.csh
```

bzw.

```
source /home/pdv/info4/mx/lib/mxenv.sh
```

einfügt. In dieser Datei wird **~info4/mx/bin** an das Ende von **PATH** gehängt und zusätzlich noch zwei alias-Definitionen vorgenommen: **mxmake** und **mxem**. Das alias **mxem** startet den Editor **xemacs** unter X-Windows in einen 123 Zeichen breiten Fenster. Das source-Kommando kann natürlich auch von Hand am Prompt der Shell eingegeben werden.

3.2 Kopieren von Dateien

Im Verzeichnis **~info4/info4/aufgabe1a** befinden sich bereits die Dateien **Makefile**, **LED.h**, **LED.S** und **Main.o**. Ins eigene Lösungsverzeichnis brauchen nur die Dateien **Makefile** und **LED.S** kopiert zu werden. Das **Makefile** steuert den Übersetzungsvorgang, und **LED.S** wird um die Lösung ergänzt. Alle anderen Dateien findet **mxmake** automatisch durch die Angaben im **Makefile**. Alle Module, die als Umgebung der Aufgabenlösung dienen sind in die Objektdatei **Main.o** kompiliert und werden zur programmierten Assembler-Prozedur dazugebunden.

4 Wenn etwas schiefgeht

Normalerweise erhaltet Ihr nach der Beendung Eures Assemblerprogramms eine entsprechende Meldung. Das Fenster mit den Ausgaben des *mx* schließt sich, sobald Ihr eine beliebige Taste drückt. Manchmal (z. B. wenn Ihr eine Endlosschleife programmiert habt) funktioniert das nicht so einfach. In diesem Fall müßt Ihr das Ausgabefenster über den Fenster-Manager schließen, weil **mxBoot** ein Control-C nur an den *mx* weiterleitet.