E 1 Adressraum und Datentypen

Lernziele

- Einen Programm-Debugger als Entwicklungshilfsmittel kennen lernen
- Variableninhalte und Variablenadressen mit Hilfe des Debuggers bestimmen
- Allgemein die Inhalte von Prozessorregistern und Hauptspeicher mit Hilfe des Debuggers auslesen
- Die Speicherablage von C-Datentypen am Beispiel kennen lernen
- Die Ausrichtung von Datenwerten im Adressraum anhand eines Beispiels experimentell bestimmen

Übungsumgebung

- PC mit Linux
- DDD (Data Display Debugger) installiert
- Vorlagedatei (C-Quellkode): **`adrspace.c`**

Aufgaben

E 1.1 Grafischer Debugger "DDD" unter Linux.

Nachfolgende Aufgaben werden mit dem "DDD" anhand des Beispiels **`adrspace.c`** gelöst.
Vorbereitung:
(a) Beispiel übersetzen mit **`gcc adrspace.c -g -o adrspace`**
(Hinweis: Die Kommandozeilenoption **`-g`** ist zwingend, sonst Programm nicht in DDD ladbar!)
(b) DDD starten: über Programmmenü *oder* ab Kommandozeile mittels **`ddd <Enter>`**
(c) Programm **`adrspace`** mit *File->Open Program* in Debugger laden
(d) Weitere Bedienhinweise finden Sie in der *Kurzeinführung* im Anhang

E 1.2 Registerinhalte und Variablenadressen/inhalte

a) Bestimmen Sie den Codeumfang der ersten ausführbaren C-Programmanweisung in **`main()`**, d.h. der C-Anweisung **`a=0x8899aabb;`**
Vorgehen: Setzen Sie einen Haltepunkt auf diese Quellcodezeile und führen Sie das Programm bis dorthin aus (mittels "run"): Nun den Inhalt des Programmzählers (Register **`eip`**) notieren: 0x80483c1
Führen Sie die Anweisung selbst aus (mittels "step"); der grüne Zeiger des momentanen Ausführungspunkts muss auf die nächste Anweisung zeigen. Nun den Inhalt des Programmzählers notieren: 0x80483cb
Subtrahieren Sie den ersten Wert vom zweiten Wert: Dies ist die Codegröße (Anz. Byte): 0xa -> 10byte

b) Prüfen Sie die Inhalte der Variablen **`a`** und **`d`**: Entsprechen Sie den Erwartungen?
Hinweis: Zur Anzeige des Inhalts einer globalen Variablen in Quellcodefenster den Variablennamen mit der Maus markieren und anschließend die Taste "Display" anklicken. Mit der rechten Maustaste kann dann über ein Aufklappmenü die Datendarstellung noch geändert werden.

a ist initalisiert mit dem richren hex wert
d auch anhand der angegeben werte (alle werte als hex!)

E 1.3 Analyse ganzer Speicherbereiche

Der DD-Debugger erlaubt die Inspektion ganzer Hauptspeicherbereiche in Form von "memory dumps". Ein derartiger Speicherauszug wird wie folgt interpretiert:

Anzeige:

```
0x8049420   0x13 0xab 0x4b 0xa0 0x00 0x00 0x12 0x78
0x8049428   0x00 0xff 0xfa 0x10 0x13 0x34 0x77 0x9a
0x8049430   0x34
```

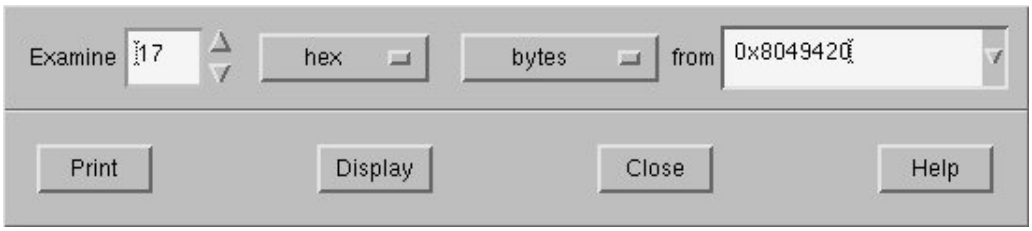
Ganz links steht die hexadezimale Speicheradresse (= 1. Spalte). Rechts davon das damit adressierte Byte sowie die Inhalte der 7 nächsthöheren Speicheradressen. Damit entspricht obige Anzeige folgendem Bild:

```
Speicher-   Speicherin-
adresse     inhalt (Bytewert)

0x8049420   0x13
0x8049421   0xab
0x8049422   0x4b
0x8049423   0xa0
0x8049424   0x00
0x8049425   0x00
0x8049426   0x12
0x8049427   0x78
0x8049428   0x00
0x8049429   0xff
0x804942a   0xfa
..
```

Andere Debugger zeigen manchmal auch 16 Byte pro Zeile an sowie teilweise rechts der hexadezimalen Darstellung zusätzlich eine Interpretation der Speicherinhalte als ASCII-kodierte Textzeichen.

Obigen *memory dump* erhält man im DDD übrigens, wenn man im Dialogfenster "*DDD: examine memory*" folgende Einstellungen verwendet (natürlich stimmen die Speicherinhalte nicht unbedingt überein):



a) Prüfen Sie mittels eines *memory dump* (Anwählen mit: *Data->memory*; es erscheint erst obiger Dialog) die Inhalte der Variablen **`d`**.
Hinweis:
Im Feld für die Startadresse ("from") erhält man die Adresse von **`d`** durch Eingabe von **`&d`**.

Entspricht die Darstellung im Speicher Ihren Erwartungen? (Benutzt der Pentium-Prozessor eine Big- oder Little-Endian-Darstellung von Mehrbyte-Datenwerten?)

( ) Big Endian (X) Little Endian

b) Welches Vektorelement liegt an der Adresse **`&d`** im Speicher (passendes unten ankreuzen):
( ) **`d[0]`** ( ) **`d[7]`** (X) anderes, nämlich: nur ein teil von d[0] -> der zweite Teil von 1122 -> 22

0x804a060 <d> 0x22 0x11 0x00 0x00.....

E 1.4 Ausrichtung von Daten im Adressraum

Um die Zugriffsgeschwindigkeit auf Datenwerte zu optimieren, werden Variablen oder auch Komponenten eines **`struct`** im Adressraum nach bestimmten Regeln ausgerichtet (alignment rules).
Zum Beispiel bedeutet eine Ausrichtung auf eine sog. 2-Byte-Grenze (align 2), dass die Hauptspeicheradresse ohne Rest durch 2 teilbar ist.
Für eine 4-Byte-Grenze (align 4) müsste die Hauptspeicheradresse ohne Rest durch 4 teilbar sein.

Hinweis: Prüfen Sie dies nach, indem Sie z.B. die konkrete Speicherbelegung des structs **`x`** prüfen: Auf welcher Art von Adressen liegen die Komponenten **`u,v,w`** (bzw. durch was sind diese Adressen ohne Rest teilbar)?

Welche Ausrichtungsregel benutzt der GNU-C/C++-Compiler (gcc): .....

E 1.5 Größe von C-Variablen

Mit Hilfe der Funktion **`sizeof()`** kann die Größe einer Variablen bzw. eines Datentyps in C bestimmt werden. Die zurückgegebene Anzahl ist die Anzahl von Byte, die im Speicher belegt werden.
Immer wenn die exakte Größe einer Variablen im Speicher benötigt wrir, sollte diese Größe mittels **`sizeof()`** bestimmt werden. Damit wird eine problemlose Portierung möglich, da bei der Erstellung des Quellcodes keine Annahmen über die Größen einzelner Datrentypen im Speicher gemacht werden müssen.
Im Beispielprogramm finden Sie mehrere Aufrufe von **`sizeof()`**:

a) Ermitteln Sie die Größen in Byte der folgenden Datenwerte des Beispielprogramms
(zur Anzeige von "**`res`**" verwenden Sie am besten: *Data-> Display Local Variables*):
`a`: 4
`x`: 12
`s`: 4
`*s`: 1
`"Hallo Welt!"`: 12

b) Wieso entspricht die für **`*s`** zurückgegebene Größe nicht der Länge des Strings?

Es wird die grösse des pointers zurückgegeben

c) Wieso ist die Länge von **`"Hallo Welt!"`** um 1 größer als die Anzahl Textzeichen zwischen den Anführungszeichen?

Es wird nicht die Anzahl Buchstaben zurückgeben, sondern die Grösse des Arrays

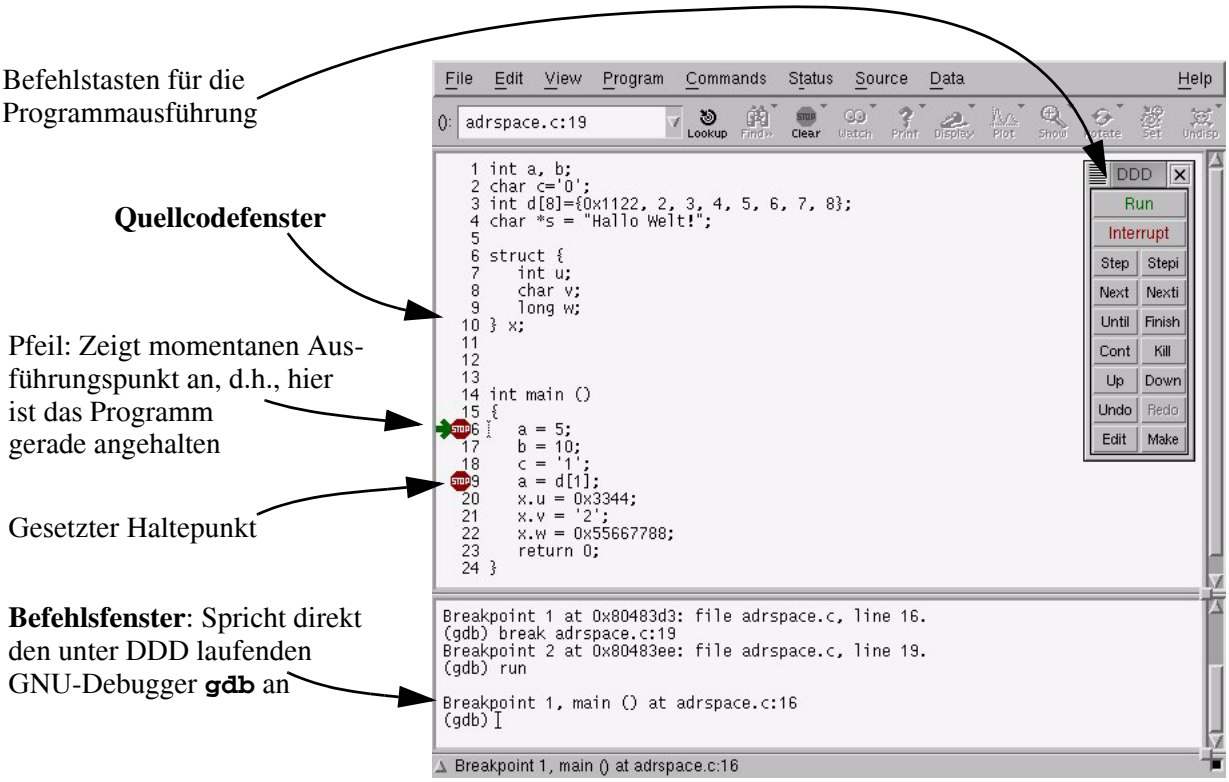
Anhang: Kurzeinführung in DDD

Der Debugger "DDD" ist ein frei erhältliches Dienstprogramm, das dem GNU-Debugger **gdb** ein grafisches Frontend verleiht. Damit können mit dem GNU-C/C++-Compiler übersetzte Programme komfortabel ausgetestet werden.

Die Möglichkeiten umfassen unter anderem Folgendes:

- Haltepunkte im Programm setzen (auf C-Quellcodezeilen), Programm bis dorthin ausführen.
- Programm im Schrittbetrieb ausführen:
  - mittels "step" wird die nächste C-Hochsprachzeile ausgeführt
  - mittels "stepi" wird der nächste, einzelne Maschinenbefehl ausgeführt
  Schrittbetrieb heißt: Programm läuft an und wird nach einem Schritt wieder angehalten.
- Bei angehaltenem Programm die momentanen Werte von Variablen, momentanen Registerinhalte und Speicherinhalte (z.B. Stapel) anzeigen.

Bild nach dem Starten des DDD und Laden eines C-Programms:



Anzeige von Inhalten der Prozessorregister (Fenster einblenden mit: *Status->Registers)*