AVR-Simulator

Das Programm soll Anfängern unter Linux helfen, die ersten Schritte in der Assemblerprogrammierung für AVR zu bewältigen.

Hintergrund

Das Programm entstand vor dem Hintergrund, das AVR-Studio 4 unter WINE nur sehr unzuverlässig funktioniert. Die Kombination aus "simulavr" + "avr-gdb" und gegebenenfalls "DDD" funktioniert gut, für Projekte die überwiegend in C erstellt sind. Kleine Projekte, die ausschließlich in Assembler programmiert sind, lassen sich nur auf Binärebene debuggen, was wohl auch daran liegt, das der gdb auf die Debug-Informationen in der ELF-Datei angewiesen ist, welche aber nicht alle Assembler erzeugen.

Zweck

Simuliert wird primär der Prozessorkern (CPU). Die zukünftige Implementierung allgemeiner Peripherie ist aber denkbar. Hier kommen primär Komponenten in Frage, die in vielen Bauelementen in gleicher Weise implementiert sind. Bis dahin werden die I/O-Register wie RAM behandelt, können also beschrieben, und auch wieder zurück gelesen werden.

Gelesen werden Objektdateien (.obj), die sowohl vom proprietären AVRASM2, dem freien avra, sowie dem ebenfalls freien Universalassembler "AS" von Alfred Arnold geschrieben werden können. Diese enthalten neben dem eigentlichen Programm, auch Informationen, zu den ursprünglichen Quelldateien.

Ebenfalls ist es auch möglich, Intel-Hexdateien (.hex) einzulesen. Da hier aber keine Debug-Informationen vorliegen, kann der Programmablauf hier nur anhand der Disassemblerausgabe verfolgt werden.

Installation

Der Simulator wird im Quelltext bereitgestellt und benutzt das FLTK-Framework.

Linux

Bevor das Programm compiliert werden kann, ist sicherzustellen, das FLTK installiert ist:

```
sudo apt-get install libfltk1.3-dev
sudo apt-get install libx11-dev
```

Im Verzeichnis "src" befindet sich das Shellscript "c", welches die Compilierung durchführt.

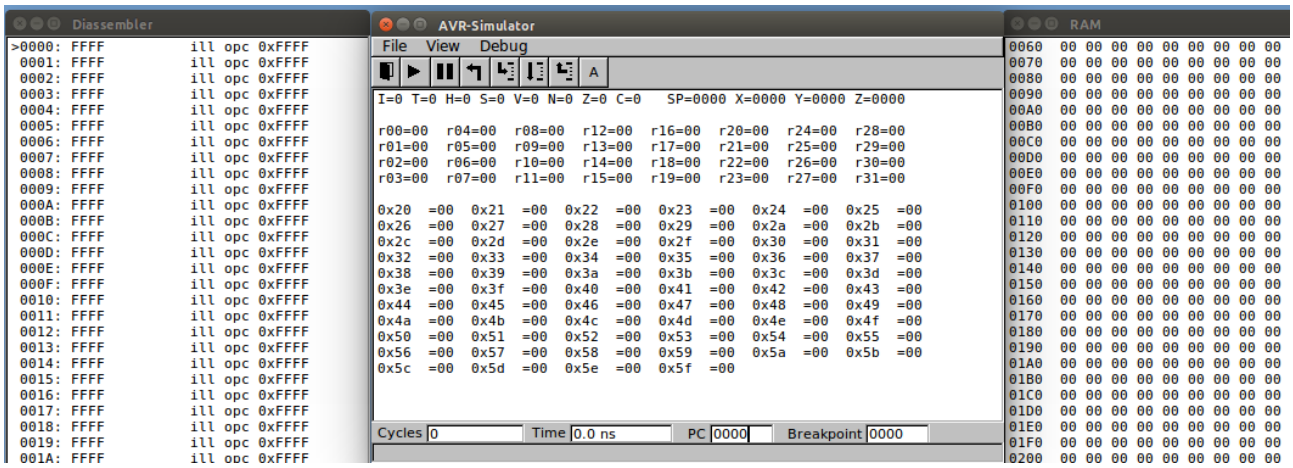
Die entstandene Programmdatei "avrsim" kann dann bei Bedarf, an einen geeigneteren Ort verschoben werden.

MS-Windows

Für win32, wird ein kompiliertes Paket bereitgestellt, welches die erforderlichen MinGW-Bibliotheken, das statisch gelinkte Programm, sowie erforderliches Zubehör enthält.

Einrichtung

Wird das Programm das erste Mal gestartet, liegen noch keine Informationen über die Art des zu simulierenden Prozessors vor. Für die I/O-Ports können daher keine Namen angezeigt werden.



Diese Informationen werden aus den Standard-Includedateien des AVRASM2 gelesen (File->Load IoDefs). Für die gängigsten Bausteine, liegen sie im Verzeichnis src/Appnotes oder Win32bin/Appnotes bei.

Aus diesen Dateien wird insbesondere die Lage des Statusregister, der Stackpointer, die Speicherlimits, sowie die Benennung der I/O-Register bezogen.

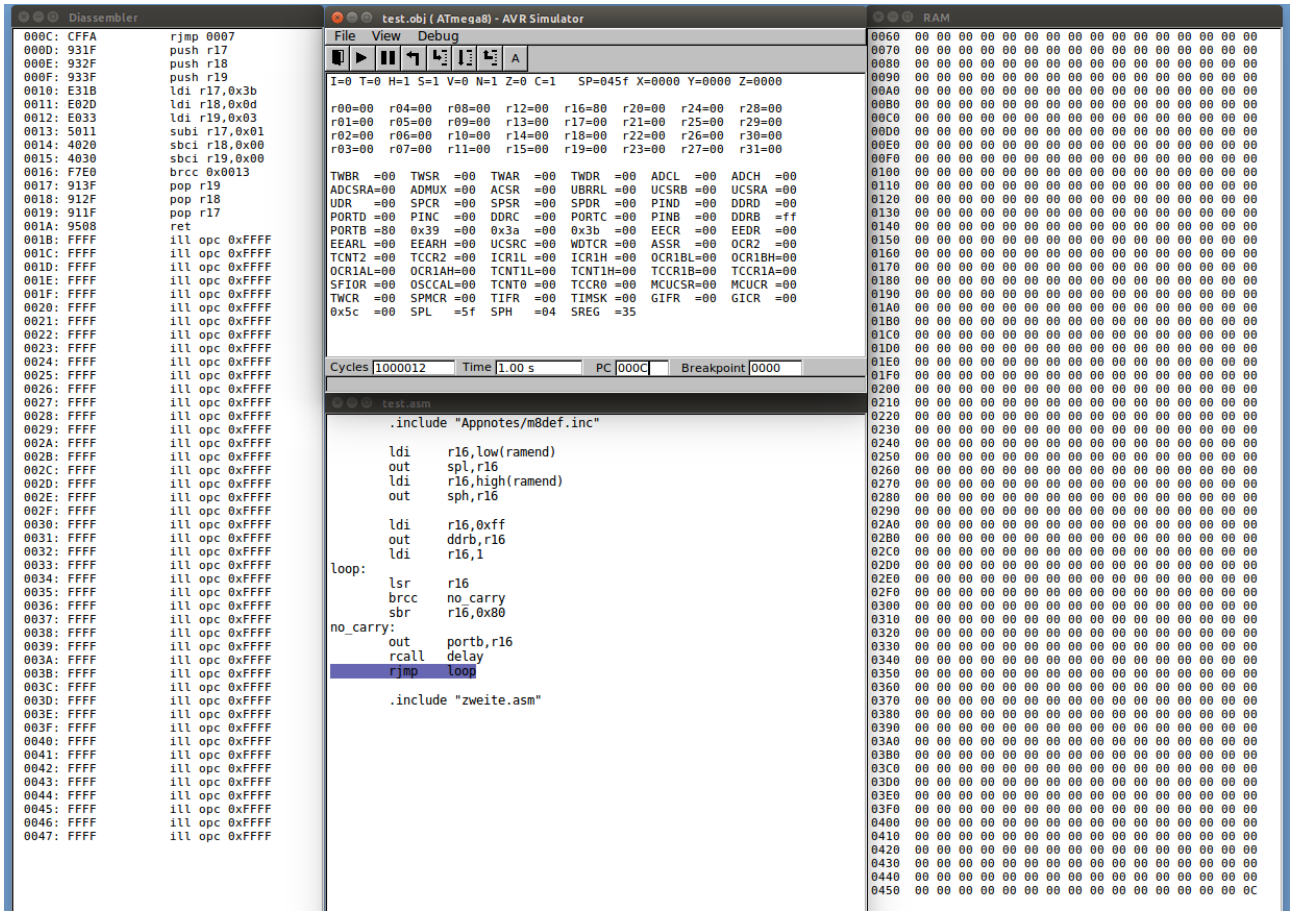
Benutzung

Die Benutzeroberfläche besteht aus dem Hauptfenster, sowie einzeln ein/ausblendbaren Fenstern für Disassemblerausgabe, Quelltext, RAM und einem Terminalfenster. Letzteres hat noch experimentellen Charakter und zeigt die Zeichen an, die in das SBUF-Register geschrieben werden.

Die zu simulierende Programmdatei wird mit "File->Load Code File" in den virtuellen Flash geladen.

Die Oberfläche für die Benutzereingaben, orientiert sich für die bisher verfügbaren Funktionen am AVR-Studio 4.

An der Ausgabe gibt es noch Etwas zu tun, bisher werden alle Informationen in Listenform in Textfenster ausgegeben. Dies hat einige Nachteile, so muss der gesamte Hexdump des RAM neu ausgegeben werden, auch wenn sich nur hinten am Stack ein Byte geändert hat, was auch zu eigenmächtigen Veränderungen der Rollbalken führen würde. Hier ist also die nächste Überarbeitung zu erwarten.



Run

Das Programm läuft im Hintergrund, ohne Aktualisierung der Anzeigen, mit voller Geschwindigkeit, bis zum gegebenenfalls gesetzten Haltepunkt.

Break

Damit kann das Programm abgebrochen und inspiziert werden.

Reset

Ein gegebenenfalls laufendes Programm wird abgebrochen, der Befehlszähler und die Stoppuhr zurückgesetzt und das Programm neu in den Flash geladen (für den Fall, dass es inzwischen geändert worden sein sollte).

Step Into

Es wird ein Prozessorbefehl ausgeführt. Sollte es sich dabei um einen Call handeln, ist der nächste Befehl, der erste der Subroutine.

Step Over

Sollte es sich hier um einen Call handeln, wird die gesamte Subroutine ausgeführt. Falls das Programm in der Subroutine "hängen" sollte, kann die mit Break unterbrochen und inspiziert werden.

Step Out

Hier werden alle Befehle ausgeführt, bis die Routine mit RET beendet wird ausgeführt.

Auto Step

Das Programm läuft, bis es mit Break unterbrochen wird. Dabei wird der Prozessorstatus auf dem Bildschirm angezeigt, was ein Vielfaches der Zeit benötigt ;-)

Die Stoppuhr

Im Eingabefeld "Cycles" wird ein Zähler der ausgeführten Taktzyklen geführt. Dieser kann bei Bedarf manuell "genullt", oder wie gewünscht geändert werden. Von Diesem abhängig, ist die Anzeige "Time". Diese setzt voraus, das unter "View->Settings" die richtige Taktfrequenz (CPU Speed, in Hz) eingestellt ist.

Breakpoint

Bisher ist ein Haltepunkt einstellbar (wird sich hoffentlich bald ändern). Ein eingestellter Wert von 0 kommt nur zum Tragen, wenn der PC überläuft, oder ein Sprungbefehl auf diese Adresse stattfindet.

Include Suchpfad

Je nach Gestaltung der Quellen, enthält die Objektdatei unter Umständen nicht den vollständigen Pfad zur entsprechenden Quelldatei. Falls eine Datei ohne Pfad angegeben ist, wird als Erstes versucht, sie im Verzeichnis der Objektdatei zu finden.

Sollten die Includedateien "anderswo" stehen und dieser Pfad dem Assembler per Kommandozeilenargument mitgegeben werden, kann der Simulator unter "View->Settings", Includepath darüber unterrichtet werden. Hier können bis zu 10 Verzeichnisse, durch Doppelpunkt voneinander getrennt, angegeben werden.

Starter Erzeugen

Benutzer von Gnome (und kompatiblen) Desktops, können mit "File->Create Starter" eine Desktopdatei in ihrem persönlichem Applications-Verzeichnis erzeugen. Damit lässt sich dann die Erweiterung ".obj" mit dem Programm verknüpfen, und die Dateien lassen sich dann mit Doppelklick öffnen.