Embedded Systems: Aufgabenblatt 5

Prof. M. König

**Praktikum 5 (10 Punkte) - Assembler**

Die Abgabe der Lösungen erfolgt im Ilias bis zum angegebenen Abgabezeitpunkt. Die Besprechung und Bewertung der **eingereichten** Lösungen erfolgt am folgenden Praktikumstermin.

**Vorbereitung**

Infomaterial finden Sie u.a. im Ilias:

* • ARMv7-M Architecture Reference Manual, Kapitel A4 („The ARMv7-M Instruction Set“) und A5 („Thumb Instruction Set Encoding“)
* • Cortex-M4 Devices Generic User Guide, Kapitel 3 „The Cortex-M4 Instruction Set“

**Praktikum**

**Aufgabe 1** *(4 Punkte)*

Arm-none-ebabi-gcc Optionen:

-o0:

Mit dem o0 Flag wird die niedrigste Stufe der Codeoptimierung ausgeführt. Dabei wird die Kompilierungszeit beschleunigt und das Debuggen ermöglicht.

-o2:

O2 optimiert sowohl die Kompilierzeit als auch die Performanz des erstellten Codes.

-os:

Der os Flag sorgt dafür, dass der Code schlank bleibt. Dabei aktiviert er alle o2 Flags, die die Codegröße nicht beeinflussen.

arm-none-eabi-size blink.elf:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| text | data | bss | dec | hex | filename |
| 888 | 0 | 260 | 1148 | 47c | blink.elf |

Disassembly:

In der ersten Spalte steht der Wert, auf den der Programmcounter zeigt, wenn der Befehl ausgeführt wird. Die zweite Spalte beinhaltet den Befehl in hexadezimal und in der 3. Spalte steht der Befehl nochmal in Assembler.

Assembler Befehle:

|  |  |
| --- | --- |
| 2ae: 4b0d  ldr r3, [pc, #52] | Die Speicheradresse auf die PC zeigt wird um 52 Bytes erhöht und der Inhalt dieser Adresse wir im Register r3 geladen. |
| 2b0: 2208  movs r2, #8 | Lädt den Wert 8 in das Register r2 |
| 2b2: 601a  str r2, [r3, #0] | Speichert den Inhalt des Registers r2 in das Register r3. |
| 2b4: 4b0c  ldr r3, [pc, #48] | Die Speicheradresse auf die PC zeigt wird um 48 Bytes erhöht und der Inhalt dieser Adresse wir im Register r3 geladen. |
| 2b6: 2208  movs r2, #8 | Lädt den Wert 8 in das Register r2 |
| 2b8: 601a  str r2, [r3, #0] | Speichert den Inhalt des Registers r2 in das Register r3. |
| 2ba: 4a0c  ldr r2, [pc, #48] | Die Speicheradresse auf die PC zeigt wird um 48 Bytes erhöht und der Inhalt dieser Adresse wir im Register r2 geladen. |
| 2bc: 4b0b  ldr r3, [pc, #44] | Die Speicheradresse auf die PC zeigt wird um 44 Bytes erhöht und der Inhalt dieser Adresse wir im Register r3 geladen. |
| 2be: 681b  ldr r3, [r3, #0] | Der Wert, der in der Speicheradresse, auf die r3 zeigt wird in r3 geladen. |
| 2c0: f043 0308  orr.w r3, r3, #8 | Verknüpft r3 mit 8 und schreibt das Ergebnis in r3, Das heißt das 4. Bit von r3 wird auf 1 gesetzt. |
| 2c4: 6013  str r3, [r2, #0] | Speichert den Inhalt des Registers r3 in das Register r2. |
| 2c6: f7ff ffd1  Bl 26c <delay> | Springt zum ersten Befehl in <delay>. |
| 2ca: 4a08  ldr r2, [pc, #32] | Die Speicheradresse auf die PC zeigt wird um 32 Bytes erhöht und der Inhalt dieser Adresse wir im Register r2 geladen. |
| 2cc: 4b07  ldr r3, [pc, #28] | Die Speicheradresse auf die PC zeigt wird um 28 Bytes erhöht und der Inhalt dieser Adresse wir im Register r3 geladen. |
| 2ce: 681b  ldr r3, [r3, #0] | Der Wert, der in der Speicheradresse ist, auf die r3 zeigt wird in r3 geladen. |
| 2d0: f023 0308  bic.w r3, r3, #8 | Setzt das 4. Bit von r3 auf 0, indem r3 mit dem Kompliment von 8 AND verknüpft wird. |
| 2d4: 6013  str r3, [r2, #0] | Speichert den Inhalt des Registers r3 in das Register r2. |
| 2d6: f7ff ffc9  bl 26c <delay> | Springt zum ersten Befehl in <delay>. |
| 2da: e7ee  b.n 2ba <main+0x1e> |  |

**Aufgabe 2:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Text | Data | Bss | Dec | Hex | Filename |
| 844 | 0 | 260 | 1104 | 450 | Blink.elf |

**Disassembly Unterschiede:**

Durch das Benutzen der 2. Optimierungsstufe wurde der Text Anteil reduziert, der bss Anteil hingegen ist gleichgeblieben.

Beim Öffnen der .asm Datei fällt als erstes auf, dass „blink2“ 24 Zeilen weniger hat als „blink0“. Bei genauerer Betrachtung kann man sehen das die Funktion „delay“ rausgenommen wurde und in die Main eingefügt wurde. Die doppelten Variablen wurden dadurch entfernt. Außerdem wurden alle push Befehle rausgenommen. Im ersten Schritt der Main werden alle Variablen in jeweils ein Register geladen und anschließend wird nur noch mit diesen Registern gearbeitet so, dass an ldr befehle gespart wurde (von 19 auf 15).

Durch das Optimieren des Programms läuft die for schleife in delay schneller durch, was zur Folge hat, dass die LED schneller blinkt.

**Aufgabe 3:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Text | Data | Bss | Dec | Hex | Filename |
| 812 | 0 | 260 | 1072 | 430 | Blink.elf |

Durch das Entfernen des Schlüsselworts „volatile“ hat sich der Programm Code um einiges verkürzt. Durch das Wegfallen des Schlüsselworts kann der Compiler die Variable weg optimieren, was zur folge hat das die for schleife zu einer endlosschleife wird.