Untersuchung des von gcc 4.1.1 produzierten Assembler-Codes.

Ab Optimierungsstufe –O1 bis einschließlich Optimierungsstufe –O3 und –Os werden Funktionen ohne globale Variablen, ohne Eingabeparameter und ohne Rückgabewerte wegrationalisiert. Es verbleiben nur noch der Einsprung und der Rücksprung, ganz gleich ob nun der Code für den ARM 7, MIPS 4kc oder PowerPC 405 betrachtet wird.

Ebenfalls fällt auf, dass die Optimierung auch die Reihenfolge der Funktionen verändert.

Das C-Schlüsselwort „register“ hat definitiv nur auf unoptimierten Code eine (sichtbare) Auswirkung. Der Code wird tatsächlich dahin geändert, dass zur Verfügung stehende Register benutzt werden. Dieses Verhalten ist aber kein Fehler.

Es lässt sich bei verschiedenen Funktionen feststellen, dass alle Optimierungsstufen denselben Assembler-Code liefern. Allerdings gibt es auch Funktionen bei denen nicht alle vier Optimierungsstufen gleichen Code besitzen.

Ob nun die Option –Os tatsächlich besonders Speicher optimierten Code liefert, kann ich aus beiden oben genannten Gründen nicht verifizieren.

Das Verhalten bei Interrupts konnte ich nicht analysieren, da der gcc – so meine Recherche – nur Interrupts bei ARM aber nicht bei MIPS oder PowerPC implementiert habe. Dieses allerdings stimmt mich nachdenklich. Fehlende Interrupt-Unterstützung kann ich mir einfach nicht erklären, zumal es sich bei den Prozessoren nicht um unbekannte Prozessoren handelt. Falls allerdings der gcc Interrupt-Programmierung für PowerPC und MIPS unterstützt, so ist die Dokumentation dieser Fähigkeit sehr schlecht (nicht vorhanden oder sehr gut versteckt).

Fallstudie:
Der C-Code:

```
/* Globale Variable für den vergleichenden Test des Assemblercodes: Funktion mit
lokalen gegen Funktion mit globalen Variable  */

int a_global;

void test_schleife1()
{
    int i, a;

    for(i = 0, a = 0; i < 10; i++ )
    {
       a += i*i;
    }
}

/** Modifiziert die globale Integer-Variablen a_global*/
void test_global_schleife1()
{
    int i;

    for(i = 0, a_global = 0; i < 10; i++ )
    {
       a_global += i*i;
    }
/* Merke: a_global hat nach der Schleife den Wert 285 */
}
```

ARM 7 tdmi — Assembleroutput des gcc 4.1.1 ohne Optimierung

```
        .align 2
        .global      test_schleife1
        .type  test_schleife1, %function
test_schleife1:
        @ args = 0, pretend = 0, frame = 8
        @ frame_needed = 1, uses_anonymous_args = 0
        @ basic block 1
        mov    ip, sp @,
        stmfd  sp!, {fp, ip, lr, pc}     @,
        sub    fp, ip, #4   @,,
        sub    sp, sp, #8   @,,
        mov    r3, #0 @ tmp102,
        str    r3, [fp, #-20]      @ tmp102, i
        mov    r3, #0 @ tmp103,
        str    r3, [fp, #-16]      @ tmp103, a
        b      .L60   @
.L61:
        @ basic block 2
        ldr    r2, [fp, #-20]      @ i, i
        ldr    r3, [fp, #-20]      @ i, i
        mul    r2, r3, r2   @ D.2462, i, i
        ldr    r3, [fp, #-16]      @ a, a
        add    r3, r3, r2   @ tmp107, a, D.2462
        str    r3, [fp, #-16]      @ tmp107, a
        ldr    r3, [fp, #-20]      @ i, i
        add    r3, r3, #1   @ tmp109, i,
        str    r3, [fp, #-20]      @ tmp109, i
.L60:
        @ basic block 3
        ldr    r3, [fp, #-20]      @ i, i
        cmp    r3, #9 @ i,
        ble    .L61   @,
        @ basic block 7
        sub    sp, fp, #12
        ldmfd  sp, {fp, sp, pc}
        .size  test_schleife1, .-test_schleife1
```

```
        .align 2
        .global      test_global_schleife1
        .type  test_global_schleife1, %function
test_global_schleife1:
        @ args = 0, pretend = 0, frame = 4
        @ frame_needed = 1, uses_anonymous_args = 0
        @ basic block 1
        mov    ip, sp @,
        stmfd  sp!, {fp, ip, lr, pc}     @,
        sub    fp, ip, #4   @,,
        sub    sp, sp, #4   @,,
        mov    r3, #0 @ tmp104,
        str    r3, [fp, #-16]      @ tmp104, i
        ldr    r2, .L69     @ tmp105,
        mov    r3, #0 @ tmp106,
        str    r3, [r2, #0] @ tmp106, a_global
        b      .L65   @
.L66:
        @ basic block 2
        ldr    r2, [fp, #-16]      @ i, i
        ldr    r3, [fp, #-16]      @ i, i
        mul    r2, r3, r2   @ D.2469, i, i
        ldr    r3, .L69     @ tmp109,
        ldr    r3, [r3, #0] @ a_global.3, a_global
        add    r2, r2, r3   @ D.2471, D.2469, a_global.3
        ldr    r3, .L69     @ tmp110,
        str    r2, [r3, #0] @ D.2471, a_global
        ldr    r3, [fp, #-16]      @ i, i
        add    r3, r3, #1   @ tmp112, i,
        str    r3, [fp, #-16]      @ tmp112, i
.L65:
        @ basic block 3
        ldr    r3, [fp, #-16]      @ i, i
        cmp    r3, #9 @ i,
        ble    .L66   @,
        @ basic block 7
        ldmfd  sp, {r3, fp, sp, pc}      @
.L70:
        .align 2
.L69:
        .word  a_global
        .size  test_global_schleife1, .-test_global_schleife1
```

rot wurde die Abfrage des Schleifenkopf markiert: Es wird zunächst der aktuelle Wert der Variablen i (Zugriff über den Framepointer berechnet) nach r3 geladen, dann wird r3 mit dem Wert 9 verglichen. Ist der Wert in r3 kleiner, so wird in die Schleife gesprungen.
blau wurde der Inhalt des Schleifenkörpers markiert: Hier unterscheiden sich die Variante mit der globalen Variable ganz leicht von der ohne globale Variable. Zunächst wird wieder der Inhalt von i nach r2 und r3 geladen. Man beachte in r3 ist schon der aktuelle Wert von i. Nun wird r2 mit r3 multipliziert und das Ergebnis in r2 abgespeichert. Nun wird der aktuelle Wert von a bzw a_global nach r3 geladen. Es folgt die Berechnung von $a+i^2$. Und hier ist der Unterschied zufinden.
Lokale Variante: r2 ($=i^2$) wird mit r3 (=a) addiert und das Ergebnis bei r3 abgelegt. Nun wird r3 über den Framepointer an die Speicherstelle von a zurückgeschrieben. Anschließend wird der aktuelle Wert von i, dem Schleifenzähler, nach r3 geladen dort inkrementiert und zurück nach i geschrieben.
Globale Variante: Nach den beiden Zeilen

„
```
ldr    r3, .L69     @ tmp109,
ldr    r3, [r3, #0] @ a_global.3, a_global
```
“

befindet sich der aktuelle Wert der globalen Variable a_global in r3. Nun wird r2 ($=i^2$) mit r3 (=a_global) addiert und das Ergebnis bei r3 abgelegt.
Nun folgt das Zurückschreiben des neuen Wertes für a_global, mittels:

„
```
ldr    r3, .L69     @ tmp110,
str    r2, [r3, #0] @ D.2471, a_global
```
“

Anschließend wird der aktuelle Wert von i, dem Schleifenzähler, nach r3 geladen dort inkrementiert und zurück nach i geschrieben.

Assembleroutput des gcc 4.1.1 erste Optimierung

```
        .align 2
        .global      test_schleife1
        .type  test_schleife1, %function
test_schleife1:
        @ args = 0, pretend = 0, frame = 0
        @ frame_needed = 0, uses_anonymous_args = 0
        @ link register save eliminated.
        @ basic block 0
        @ lr needed for prologue  @
        bx     lr    @
        .size  test_schleife1, .-test_schleife1
```

```
        .align 2
        .global      test_global_schleife1
        .type  test_global_schleife1, %function
test_global_schleife1:
        @ args = 0, pretend = 0, frame = 0
        @ frame_needed = 0, uses_anonymous_args = 0
        @ link register save eliminated.
        @ basic block 0
        @ lr needed for prologue  @
        mov    r2, #0 @ tmp106,
        ldr    r3, .L47     @ tmp105,
        str    r2, [r3, #0] @ tmp106, a_global
        mov    r3, r2 @ i, a_global_lsm.432
.L42:
        @ basic block 1
        mla    r2, r3, r3, r2      @ a_global_lsm.432, i, i, a_global_lsm.432
        add    r3, r3, #1   @ i, i,
        cmp    r3, #10      @ i,
        bne    .L42  @,
        @ basic block 2
        ldr    r3, .L47     @ tmp108,
        str    r2, [r3, #0] @ a_global_lsm.432, a_global
        bx     lr    @
.L48:
        .align 2
.L47:
        .word  a_global
        .size  test_global_schleife1, .-test_global_schleife1
```

Offensichtlich ist die Funktion der lokalen Variante wegoptimiert worden. Der Schleifenkörper in der globalen Variante besteht genau aus zwei Befehlen, dem `mla r2, r3, r3, r2` (dies realisiert r2 = r3*r3 + r2) sowie dem `add r3, r3, #1 @ i, i,` dies macht den Inkrement des Schleifenzählers.

Assembleroutput des gcc 4.1.1 zweite Optimierung

```
        .align 2
        .global      test_schleife1
        .type  test_schleife1, %function
test_schleife1:
        @ args = 0, pretend = 0, frame = 0
        @ frame_needed = 0, uses_anonymous_args = 0
        @ link register save eliminated.
        @ basic block 0
        @ lr needed for prologue  @
        bx     lr     @
        .size  test_schleife1, .-test_schleife1
```

```
        .align 2
        .global      test_global_schleife1
        .type  test_global_schleife1, %function
test_global_schleife1:
        @ args = 0, pretend = 0, frame = 0
        @ frame_needed = 0, uses_anonymous_args = 0
        @ link register save eliminated.
        @ basic block 0
        ldr    r1, .L49     @ tmp109,
        mov    r3, #0 @ tmp106,
        str    r3, [r1, #0] @ tmp106, a_global
        @ lr needed for prologue  @
        mov    r2, r3 @ a_global_lsm.435, tmp106
.L44:
        @ basic block 1
        mla    r2, r3, r3, r2      @ a_global_lsm.435, i, i, a_global_lsm.435
        add    r3, r3, #1   @ i, i,
        cmp    r3, #10      @ i,
        bne    .L44  @,
        @ basic block 2
        str    r2, [r1, #0] @ a_global_lsm.435, a_global
        bx     lr     @
.L50:
        .align 2
.L49:
        .word  a_global
        .size  test_global_schleife1, .-test_global_schleife1
```

Hier sieht man im Bereich vor der Schleife, dass Befehlsreihenfolge verändert wurde.

Assembleroutput des gcc 4.1.1 dritte Optimierung

```
        .align 2
        .global      test_schleife1
        .type  test_schleife1, %function
test_schleife1:
        @ args = 0, pretend = 0, frame = 0
        @ frame_needed = 0, uses_anonymous_args = 0
        @ link register save eliminated.
        @ basic block 0
        @ lr needed for prologue  @
        bx     lr    @
        .size  test_schleife1, .-test_schleife1
```

```
        .align 2
        .global      test_global_schleife1
        .type  test_global_schleife1, %function
test_global_schleife1:
        @ args = 0, pretend = 0, frame = 0
        @ frame_needed = 0, uses_anonymous_args = 0
        @ link register save eliminated.
        @ basic block 0
        mov    r3, #284     @ tmp103,
        ldr    r2, .L45     @ tmp101,
        add    r3, r3, #1   @ tmp102, tmp103,
        @ lr needed for prologue  @
        str    r3, [r2, #0] @ tmp102, a_global
        bx     lr    @
.L46:
        .align 2
.L45:
        .word  a_global
        .size  test_global_schleife1, .-test_global_schleife1
```

Die Optimierung 3 scheint fehlerhaft zu sein, **ist sie aber nicht**: Die Schleife fehlt, die Multiplikation auch, allerdings wird eine temporäre Variable (r3) zunächst auf 284 gesetzt und anschließend inkrementiert, so dass diese den Wert 285 besitzt. Dieser wird dann in der globalen Variablen abgespeichert.

Assembleroutput des gcc 4.1.1 Speicher-Optimierung

```
        .align 2
        .global      test_schleife1
        .type  test_schleife1, %function
test_schleife1:
        @ args = 0, pretend = 0, frame = 0
        @ frame_needed = 0, uses_anonymous_args = 0
        @ link register save eliminated.
        @ basic block 0
        @ lr needed for prologue  @
        bx     lr     @
        .size  test_schleife1, .-test_schleife1
```

```
        .align 2
        .global      test_global_schleife1
        .type  test_global_schleife1, %function
test_global_schleife1:
        @ args = 0, pretend = 0, frame = 0
        @ frame_needed = 0, uses_anonymous_args = 0
        @ link register save eliminated.
        @ basic block 0
        ldr    r3, .L48     @ tmp105,
        mov    r2, #0 @ tmp106,
        str    r2, [r3, #0] @ tmp106, a_global
        @ lr needed for prologue  @
        mov    r3, r2 @ i, a_global_lsm.409
.L43:
        @ basic block 1
        mla    r2, r3, r3, r2      @ a_global_lsm.409, i, i, a_global_lsm.409
        add    r3, r3, #1   @ i, i,
        cmp    r3, #10      @ i,
        bne    .L43  @,
        @ basic block 2
        ldr    r3, .L48     @ tmp108,
        str    r2, [r3, #0] @ a_global_lsm.409, a_global
        bx     lr     @
.L49:
        .align 2
.L48:
        .word  a_global
        .size  test_global_schleife1, .-test_global_schleife1
```

Die Speicher-Optimierungsstufe ist sehr ähnlich der Optimierungsstufe 2, im Basic block 2 steht hier ein ldr mehr.

~~Neben der aufgezeigten fehlerhaften Optimierungsstufe 3 bei ARM, wird auch der Thumbcode fehlerhaft optimiert:~~

Die Optimierungsstufe 3 bei ARM imThumbcode sieht wie folgt aus, hier wird das Schleifenergebnis, die 285, direkt eingefügt.

```
	.align 2
	.global	test_global_schleife1
	.code 16
	.thumb_func
	.type	test_global_schleife1, %function
test_global_schleife1:
	@ basic block 0
	ldr	r2, .L58	@ tmp101,
	ldr	r3, .L58+4	@ tmp102,
	@ lr needed for prologue  @
	str	r3, [r2]	@ tmp102, a_global
	@ sp needed for prologue  @
	bx	lr
.L59:
	.align 2
.L58:
	.word	a_global
	.word	285
	.size	test_global_schleife1, .-test_global_schleife1
```

~~Aber auch bei den Prozessoren MIPS 4kc sowie PowerPC 405 ist die Optimierungsstufe 3 buggy. Hier im Folgenden zum Vergleich nun der Assembler-Code für den MIPS 4kc in allen Optimierungen und anschließend der Assemblercode für den PowerPC 405.~~

Aber auch bei den Prozessoren MIPS 4kc sowie PowerPC 405 wird in der Optimierungsstufe 3 so verfahren. Hier im Folgenden zum Vergleich nun der Assembler-Code für den MIPS 4kc in allen Optimierungen und anschließend der Assemblercode für den PowerPC 405.

Dieses Verfahren hat aber deutliche Nachteile falls die globale Variable von Interrupts verwendet werden soll. Doch dazu unten mehr.

MIPS 4kc — Assembleroutput des gcc 4.1.1 ohne Optimierung

```
        .align 2
        .globl test_schleife1
        .ent   test_schleife1
        .type  test_schleife1, @function
test_schleife1:
        .frame $fp,24,$31   # vars= 8, regs= 1/0, args= 0, gp= 8
        .mask  0x40000000,-8
        .fmask 0x00000000,0
        .set   noreorder
        .set   nomacro

         # basic block 1
        addiu  $sp,$sp,-24   #,,
        sw     $fp,16($sp)   #,
        move   $fp,$sp       #,
        sw     $0,12($fp)    #, i
        sw     $0,8($fp)     #, a
        b      $L58
        nop
         #
$L59:
         # basic block 2
        lw     $3,12($fp)    # i, i
        lw     $2,12($fp)    # i, i
        mul    $3,$3,$2      # D.2461, i, i
        lw     $2,8($fp)     # a, a
        addu   $2,$2,$3      # tmp197, a, D.2461
        sw     $2,8($fp)     # tmp197, a
        lw     $2,12($fp)    # i, i
        addiu  $2,$2,1       # tmp199, i,
        sw     $2,12($fp)    # tmp199, i
$L58:
         # basic block 3
        lw     $2,12($fp)    # i, i
        slt    $2,$2,10      # tmp201, i,
        bne    $2,$0,$L59
        nop
         #, tmp201,
         # basic block 7
        move   $sp,$fp       #,
        lw     $fp,16($sp)   #,
        addiu  $sp,$sp,24    #,,
```

```
        .align 2
        .globl test_global_schleife1
        .ent   test_global_schleife1
        .type  test_global_schleife1, @function
test_global_schleife1:
        .frame $fp,24,$31   # vars= 8, regs= 1/0, args= 0, gp= 8
        .mask  0x40000000,-8
        .fmask 0x00000000,0
        .set   noreorder
        .cpload      $25
        .set   nomacro

         # basic block 1
        addiu  $sp,$sp,-24   #,,
        sw     $fp,16($sp)   #,
        move   $fp,$sp       #,
        sw     $0,8($fp)     #, i
        lw     $2,%got(a_global)($28)     # tmp196,,
        sw     $0,0($2)      #, a_global
        b      $L63
        nop
         #
$L64:
         # basic block 2
        lw     $3,8($fp)     # i, i
        lw     $2,8($fp)     # i, i
        mul    $3,$3,$2      # D.2468, i, i
        lw     $2,%got(a_global)($28)     # tmp199,,
        lw     $2,0($2)      # a_global.3, a_global
        addu   $3,$3,$2      # D.2470, D.2468, a_global.3
        lw     $2,%got(a_global)($28)     # tmp200,,
        sw     $3,0($2)      # D.2470, a_global
        lw     $2,8($fp)     # i, i
        addiu  $2,$2,1       # tmp202, i,
        sw     $2,8($fp)     # tmp202, i
$L63:
         # basic block 3
        lw     $2,8($fp)     # i, i
        slt    $2,$2,10      # tmp204, i,
        bne    $2,$0,$L64
        nop
         #, tmp204,
```

```
        j       $31
        nop
         #
        .set    macro
        .set    reorder
        .end    test_schleife1
```

```
         # basic block 7
        move    $sp,$fp         #,
        lw      $fp,16($sp)     #,
        addiu   $sp,$sp,24      #,,
        j       $31
        nop
         #
        .set    macro
        .set    reorder
        .end    test_global_schleife1
```

MIPS 4kc — Assembleroutput des gcc 4.1.1 erste Optimierung

```
        .align 2
        .globl test_schleife1
        .ent   test_schleife1
        .type  test_schleife1, @function
test_schleife1:
        .frame $sp,0,$31    # vars= 0, regs= 0/0, args= 0, gp= 0
        .mask  0x00000000,0
        .fmask 0x00000000,0
        .set   noreorder
        .set   nomacro

         # basic block 0
        j       $31
        nop

        .set    macro
        .set    reorder
        .end    test_schleife1
```

```
        .align 2
        .globl test_global_schleife1
        .ent   test_global_schleife1
        .type  test_global_schleife1, @function
test_global_schleife1:
        .frame $sp,0,$31    # vars= 0, regs= 0/0, args= 0, gp= 0
        .mask  0x00000000,0
        .fmask 0x00000000,0
        .set   noreorder
        .cpload      $25
        .set   nomacro

         # basic block 0
        lw      $2,%got(a_global)($28)      # tmp197,,
        sw      $0,0($2)        #, a_global
        move    $2,$0   # i,
        mtlo    $0      # a_global_lsm.432,
        li      $3,10               # 0xa   # tmp199,
$L45:
         # basic block 1
        madd    $2,$2   # i, i
        addiu   $2,$2,1         # i, i,
        bne     $2,$3,$L45
        nop
         #, i, tmp199,
         # basic block 2
        lw      $2,%got(a_global)($28)      # tmp200,,
        mflo    $3      #,
        j       $31
        sw      $3,0($2)        #, a_global
```

```
	.set	macro
	.set	reorder
	.end	test_global_schleife1
```

MIPS 4kc	Assembleroutput des gcc 4.1.1 zweite Optimierung

```
	.align	2
	.globl	test_schleife1
	.ent	test_schleife1
	.type	test_schleife1, @function
test_schleife1:
	.frame	$sp,0,$31		# vars= 0, regs= 0/0, args= 0, gp= 0
	.mask	0x00000000,0
	.fmask	0x00000000,0
	.set	noreorder
	.set	nomacro

	# basic block 0
	j	$31
	nop

	.set	macro
	.set	reorder
	.end	test_schleife1
```

```
	.align	2
	.globl	test_global_schleife1
	.ent	test_global_schleife1
	.type	test_global_schleife1, @function
test_global_schleife1:
	.frame	$sp,0,$31		# vars= 0, regs= 0/0, args= 0, gp= 0
	.mask	0x00000000,0
	.fmask	0x00000000,0
	.set	noreorder
	.cpload	$25
	.set	nomacro

	# basic block 0
	lw	$4,%got(a_global)($28)	# tmp201,,
	move	$2,$0	# i,
	sw	$0,0($4)	#, a_global
	mtlo	$0	# a_global_lsm.435,
	li	$3,10			# 0xa	# tmp199,
$L51:
	# basic block 1
	madd	$2,$2	# i, i
	addiu	$2,$2,1		# i, i,
	bne	$2,$3,$L51
	nop
	#, i, tmp199,
	# basic block 2
	mflo	$2	#,
	j	$31
	sw	$2,0($4)	#, a_global

	.set	macro
	.set	reorder
	.end	test_global_schleife1
```

MIPS 4kc — Assembleroutput des gcc 4.1.1 dritte Optimierung

```
        .align 2
        .globl test_schleife1
        .ent   test_schleife1
        .type  test_schleife1, @function
test_schleife1:
        .frame $sp,0,$31    # vars= 0, regs= 0/0, args= 0, gp= 0
        .mask  0x00000000,0
        .fmask 0x00000000,0
        .set   noreorder
        .set   nomacro

        # basic block 0
        j      $31
        nop

        .set   macro
        .set   reorder
        .end   test_schleife1
```

```
        .align 2
        .globl test_global_schleife1
        .ent   test_global_schleife1
        .type  test_global_schleife1, @function
test_global_schleife1:
        .frame $sp,0,$31    # vars= 0, regs= 0/0, args= 0, gp= 0
        .mask  0x00000000,0
        .fmask 0x00000000,0
        .set   noreorder
        .cpload      $25
        .set   nomacro

        # basic block 0
        lw     $2,%got(a_global)($28)     # tmp193,,
        li     $3,285              # 0x11d       # tmp194,
        j      $31
        sw     $3,0($2)      # tmp194, a_global

        .set   macro
        .set   reorder
        .end   test_global_schleife1
```

MIPS 4kc — Assembleroutput des gcc 4.1.1 Speicher-Optimierung

```
        .align 2
        .globl test_schleife1
        .ent   test_schleife1
        .type  test_schleife1, @function
test_schleife1:
        .frame $sp,0,$31    # vars= 0, regs= 0/0, args= 0, gp= 0
        .mask  0x00000000,0
        .fmask 0x00000000,0
        .set   noreorder
        .set   nomacro

        # basic block 0
        j      $31
        nop

        .set   macro
        .set   reorder
```

```
        .align 2
        .globl test_global_schleife1
        .ent   test_global_schleife1
        .type  test_global_schleife1, @function
test_global_schleife1:
        .frame $sp,0,$31    # vars= 0, regs= 0/0, args= 0, gp= 0
        .mask  0x00000000,0
        .fmask 0x00000000,0
        .set   noreorder
        .cpload      $25
        .set   nomacro

        # basic block 0
        lw     $2,%got(a_global)($28)     # tmp197,,
        move   $3,$0  # i,
        sw     $0,0($2)      #, a_global
        move   $4,$0  # a_global_lsm.409,
```

```
        .end   test_schleife1
```

```
        li     $2,10                # 0xa   # tmp199,
$L46:
        # basic block 1
        mul    $5,$3,$3       #, i, i
        addiu  $3,$3,1        # i, i,
        bne    $3,$2,$L46     #, i, tmp199,
        addu   $4,$5,$4       # a_global_lsm.409,, a_global_lsm.409

        # basic block 2
        lw     $2,%got(a_global)($28)     # tmp200,,
        j      $31
        sw     $4,0($2)       # a_global_lsm.409, a_global

        .set   macro
        .set   reorder
        .end   test_global_schleife1
```

Diese Optimierung scheint auch buggy zu sein.

```
int R3_i = 0;

int R4 = global_a;

int R2 = 10;

int R5_iquadrat;

while (true)
{
    R5_iquadrat = R3_i * R3_i;

    R3_i = R3_i + 1;

    if (R3_i == R2) break;
}

R4 = R5_iquadrat + R4;
```

Habe ich hier den Assemblercode richtig rücktransferiert?

Bitte nachprüfen. … **Wenn** addu **nach jedem** bne ausgeführt wird, geht’s.

Das ist dasselbe wie R4 = 10*10+ R4, dies entspricht aber nicht dem gewünschten Ergebnis.

PowerPC 405 — Assembleroutput des gcc 4.1.1 ohne Optimierung

```
        .align 2
        .globl test_schleife1
        .type  test_schleife1, @function
test_schleife1:
         # basic block 1
        stwu 1,-32(1) #,,
        stw 31,28(1)  #,
        mr 31,1       #,
        li 0,0  # tmp120,
        stw 0,12(31)  # i, tmp120
        li 0,0  # tmp121,
        stw 0,8(31)   # a, tmp121
        b .L58  #
.L59:
         # basic block 2
        lwz 9,12(31)  # i, i
        lwz 0,12(31)  # i, i
        mullw 9,9,0   # D.2498, i, i
        lwz 0,8(31)   # a, a
        add 0,0,9     # tmp125, a, D.2498
        stw 0,8(31)   # a, tmp125
        lwz 9,12(31)  # i, i
        addi 0,9,1    # tmp127, i,
        stw 0,12(31)  # i, tmp127
.L58:
         # basic block 3
        lwz 0,12(31)  # i, i
        cmpwi 7,0,9   #, tmp129, i
        ble 7,.L59    #
         # basic block 7
        lwz 11,0(1)   #,
        lwz 31,-4(11) #,
        mr 1,11       #,
        blr     #
        .size  test_schleife1, .-test_schleife1
```

```
        .align 2
        .globl test_global_schleife1
        .type  test_global_schleife1, @function
test_global_schleife1:
         # basic block 1
        stwu 1,-32(1) #,,
        stw 31,28(1)  #,
        mr 31,1       #,
        li 0,0  # tmp122,
        stw 0,8(31)   # i, tmp122
        lis 9,a_global@ha    # tmp123,
        li 0,0  # tmp124,
        stw 0,a_global@l(9)  # a_global, tmp124
        b .L63  #
.L64:
         # basic block 2
        lwz 9,8(31)   # i, i
        lwz 0,8(31)   # i, i
        mullw 11,9,0  # D.2505, i, i
        lis 9,a_global@ha    # tmp127,
        lwz 0,a_global@l(9)  # a_global, a_global.3
        add 0,11,0    # D.2507, D.2505, a_global.3
        lis 9,a_global@ha    # tmp128,
        stw 0,a_global@l(9)  # a_global, D.2507
        lwz 9,8(31)   # i, i
        addi 0,9,1    # tmp130, i,
        stw 0,8(31)   # i, tmp130
.L63:
         # basic block 3
        lwz 0,8(31)   # i, i
        cmpwi 7,0,9   #, tmp132, i
        ble 7,.L64    #
         # basic block 7
        lwz 11,0(1)   #,
        lwz 31,-4(11) #,
        mr 1,11       #,
        blr     #
        .size  test_global_schleife1, .-test_global_schleife1
```

PowerPC 405 Assembleroutput des gcc 4.1.1 erste Optimierung

```
        .align 2
        .globl test_schleife1
        .type  test_schleife1, @function
test_schleife1:
        # basic block 0
        blr     #
        .size  test_schleife1, .-test_schleife1
```

```
        .align 2
        .globl test_global_schleife1
        .type  test_global_schleife1, @function
test_global_schleife1:
        # basic block 0
        li 0,0  # tmp125,
        lis 9,a_global@ha    # tmp124,
        stw 0,a_global@l(9)  # a_global, tmp125
        li 9,0  # i,
        li 11,0         # a_global_lsm.432,
        li 0,10         #,
        mtctr 0         # tmp130,
.L43:
        # basic block 1
        mullw 0,9,9    # tmp126, i, i
        add 11,11,0    # a_global_lsm.432, a_global_lsm.432, tmp126
        addi 9,9,1     # i, i,
        bdnz .L43      #
        # basic block 2
        lis 9,a_global@ha    # tmp129,
        stw 11,a_global@l(9)        # a_global, a_global_lsm.432
        blr     #
        .size  test_global_schleife1, .-test_global_schleife1
```

PowerPC 405 Assembleroutput des gcc 4.1.1 zweite Optimierung

```
        .align 2
        .globl test_schleife1
        .type  test_schleife1, @function
test_schleife1:
        # basic block 0
        blr     #
        .size  test_schleife1, .-test_schleife1
```

```
        .align 2
        .globl test_global_schleife1
        .type  test_global_schleife1, @function
test_global_schleife1:
        # basic block 0
        li 0,0  # tmp125,
        lis 10,a_global@ha   # tmp130,
        stw 0,a_global@l(10)        # a_global, tmp125
        li 0,10         #,
        mtctr 0         # tmp133,
        li 9,0  # i,
        li 11,0         # a_global_lsm.435,
.L45:
        # basic block 1
        mullw 0,9,9    # tmp126, i, i
```

```
       addi 9,9,1    # i, i,
       add 11,11,0   # a_global_lsm.435, a_global_lsm.435, tmp126
       bdnz .L45     #
        # basic block 2
       stw 11,a_global@l(10)      # a_global, a_global_lsm.435
       blr     #
       .size  test_global_schleife1, .-test_global_schleife1
```

PowerPC 405	Assembleroutput des gcc 4.1.1 dritte Optimierung

```
       .globl test_schleife1
       .type  test_schleife1, @function
test_schleife1:
        # basic block 0
       blr     #
       .size  test_schleife1, .-test_schleife1
```

```
       .align 2
       .globl test_global_schleife1
       .type  test_global_schleife1, @function
test_global_schleife1:
        # basic block 0
       li 0,285      # tmp121,
       lis 9,a_global@ha    # tmp120,
       stw 0,a_global@l(9)  # a_global, tmp121
       blr     #
       .size  test_global_schleife1, .-test_global_schleife1
```

PowerPC 405	Assembleroutput des gcc 4.1.1 Speicher-Optimierung

```
       .align 2
       .globl test_schleife1
       .type  test_schleife1, @function
test_schleife1:
        # basic block 0
       blr     #
       .size  test_schleife1, .-test_schleife1
```

```
       .align 2
       .globl test_global_schleife1
       .type  test_global_schleife1, @function
test_global_schleife1:
        # basic block 0
       li 0,0  # tmp125,
       lis 9,a_global@ha    # tmp124,
       stw 0,a_global@l(9)  # a_global, tmp125
       li 0,10       #,
       mtctr 0       # tmp132,
       li 9,0  # i,
       li 11,0       # a_global_lsm.409,
.L44:
        # basic block 1
       mullw 0,9,9   # tmp126, i, i
       addi 9,9,1    # i, i,
       add 11,11,0   # a_global_lsm.409, a_global_lsm.409, tmp126
       bdnz .L44     #
```

	# basic block 2 lis 9,a_global@ha # tmp129, stw 11,a_global@l(9) # a_global, a_global_lsm.409 blr # .size test_global_schleife1, .-test_global_schleife1

Der Vergleich mit dem entsprechenden i386 Assemblercode zeigt, dass auch bei diesem in der Optimierungsstufe 3 der Schleifencode durch das Schleifenergebnis ersetzt wird.

```
	.p2align 2,,3
.globl test_global_schleife1
	.type	test_global_schleife1, @function
test_global_schleife1:
	# basic block 0
	pushl	%ebp	#
	movl	%esp, %ebp	#,
	movl	$285, a_global	#, a_global
	leave
	ret
	.size	test_global_schleife1, .-test_global_schleife1
```

Compileraufrufe waren:

arm-unknown-linux-gnu-gcc -O<Optimierungsstufe> -Wall -pedantic -mcpu=arm7tdmi -dA -fverbose-asm -S ./Sourcecode.c
arm-unknown-linux-gnu-gcc -O<Optimierungsstufe> -Wall -pedantic -mthumb -mcpu=arm7tdmi -dA -fverbose-asm -S ./Sourcecode.c
mips-unknown-linux-gnu-gcc -O<Optimierungsstufe> -Wall -pedantic -march=4kc -dA -fverbose-asm -S ./Sourcecode.c
powerpc-unknown-linux-gnu-gcc -O<Optimierungsstufe> -Wall -pedantic -mpowerpc -mcpu=405 -dA -fverbose-asm -S ./Sourcecode.c

Ausführendes System:
2 x Pentium III (Katmai) 600 MHz, 512 MB L2
768 MB Hauptspeicher
Das System ist ein Compaq Proliant 1850R.
Gentoo-Linux, Kernelversion 2.6.16.19

Analyse eines Interrupts (gcc 4.1.1 , ARM 7 tdmi)

C-Code:

```
int irq2_Variable; /* Globale Variable für die InterruptServiceRoutine 2*/
int a_global; /* Globale Variablen für den vergleichenden Test des Assemblercodes:
Funktion mit lokalen gegen Funktion mit globalen Variablen  */

/*****************************************************************************/
/** Zählschleife */
void test_global_schleife1()
{/*Modifiziert die globale Integer-Variable a_global*/
    int i;
    for(i = 0, a_global = 0; i < 10; i++ )
    {
      a_global += i*i;
    }
}


/** Interrupt */
void __attribute__((interrupt("IRQ"))) isr_2(void)
{
      irq2_Variable = a_global;
      irq2_Variable++;
}


/*****************************************************************************/

int main()
{
      test_global_schleife1();
      return 0;
}
/*****************************************************************************/
```

Offensichtlich ist die Zählschleife dieselbe, wie in der obigen Assembler-Code-Untersuchung; zusätzlich ist der Assemblercode zu der Funktion „test_global_schleife1()“ identisch mit der vorherigen Untersuchung, daher wird hier nur noch der Assembler des Interrupts aufgeführt.

Anschließend betrachten wir die Auswirkung, die das C-Schlüsselwort „volatile“ auf die Funktion `test_global_schleife1()` hat, indem wir die globale Variable als volatile deklarieren (`volatile int a_global;`).

Anmerkung: Im Thumbcode wird keine IRQ-ServiceRoutine übersetzt, sondern als normale Funktion (in den Optimierungsstufen: 0,1,2,3,s).

ARM 7 tdmi, keine Optimierung

```
	.align 2
	.global	isr_2
	.type	isr_2, %function
isr_2:
	@ Interrupt Service Routine.
	@ args = 0, pretend = 0, frame = 0
	@ frame_needed = 1, uses_anonymous_args = 0
	@ basic block 1
	str	ip, [sp, #-4]!	@,
	mov	ip, sp @,
	stmfd	sp!, {r2, r3, fp, ip, lr, pc}	@,
	sub	fp, ip, #4	@,,
	ldr	r3, .L10	@ tmp104,
	ldr	r2, [r3, #0] @ a_global.1, a_global
	ldr	r3, .L10+4	@ tmp105,
	str	r2, [r3, #0] @ a_global.1, irq2_Variable
	ldr	r3, .L10+4	@ tmp106,
	ldr	r3, [r3, #0] @ irq2_Variable.2, irq2_Variable
	add	r2, r3, #1	@ D.1249, irq2_Variable.2,
	ldr	r3, .L10+4	@ tmp107,
	str	r2, [r3, #0] @ D.1249, irq2_Variable
	sub	sp, fp, #20
	ldmfd	sp, {r2, r3, fp, sp, lr}
	ldmfd	sp!, {ip}
	subs	pc, lr, #4
.L11:
	.align 2
.L10:
	.word	a_global
	.word	irq2_Variable
	.size	isr_2, .-isr_2
```

ARM 7 tdmi, erste Optimierung

```
	.align 2
	.global	isr_2
	.type	isr_2, %function
isr_2:
	@ Interrupt Service Routine.
	@ args = 0, pretend = 0, frame = 0
	@ frame_needed = 0, uses_anonymous_args = 0
	@ link register save eliminated.
	@ basic block 0
	stmfd	sp!, {r2, r3}@,
	@ lr needed for prologue	@
	ldr	r3, .L11	@ tmp102,
	ldr	r3, [r3, #0] @ a_global, a_global
	add	r3, r3, #1	@ tmp104, a_global,
	ldr	r2, .L11+4	@ tmp101,
	str	r3, [r2, #0] @ tmp104, irq2_Variable
	ldmfd	sp!, {r2, r3}
	subs	pc, lr, #4
.L12:
	.align 2
.L11:
	.word	a_global
	.word	irq2_Variable
	.size	isr_2, .-isr_2
```

zweite Optimierung

```
	.align 2
	.global	isr_2
	.type	isr_2, %function
isr_2:
	@ Interrupt Service Routine.
	@ args = 0, pretend = 0, frame = 0
	@ frame_needed = 0, uses_anonymous_args = 0
	@ link register save eliminated.
	@ basic block 0
```

dritte Optimierung

```
	.align 2
	.global	isr_2
	.type	isr_2, %function
isr_2:
	@ Interrupt Service Routine.
	@ args = 0, pretend = 0, frame = 0
	@ frame_needed = 0, uses_anonymous_args = 0
	@ link register save eliminated.
	@ basic block 0
```

```
        stmfd  sp!, {r2, r3}@,
        ldr    r3, .L12     @ tmp102,
        ldr    r2, [r3, #0] @ a_global, a_global
        ldr    r3, .L12+4   @ tmp101,
        add    r2, r2, #1   @ tmp104, a_global,
        @ lr needed for prologue  @
        str    r2, [r3, #0] @ tmp104, irq2_Variable
        ldmfd  sp!, {r2, r3}
        subs   pc, lr, #4
.L13:
        .align 2
.L12:
        .word  a_global
        .word  irq2_Variable
        .size  isr_2, .-isr_2
```

```
        stmfd  sp!, {r2, r3}@,
        ldr    r3, .L7      @ tmp102,
        ldr    r2, [r3, #0] @ a_global, a_global
        ldr    r3, .L7+4    @ tmp101,
        add    r2, r2, #1   @ tmp104, a_global,
        @ lr needed for prologue  @
        str    r2, [r3, #0] @ tmp104, irq2_Variable
        ldmfd  sp!, {r2, r3}
        subs   pc, lr, #4
.L8:
        .align 2
.L7:
        .word  a_global
        .word  irq2_Variable
        .size  isr_2, .-isr_2
```

Speicheroptimierung

```
        .align 2
        .global      isr_2
        .type  isr_2, %function
isr_2:
        @ Interrupt Service Routine.
        @ args = 0, pretend = 0, frame = 0
        @ frame_needed = 0, uses_anonymous_args = 0
        @ link register save eliminated.
        @ basic block 0
        stmfd  sp!, {r2, r3}@,
        ldr    r3, .L11     @ tmp102,
        ldr    r3, [r3, #0] @ a_global, a_global
        ldr    r2, .L11+4   @ tmp101,
        add    r3, r3, #1   @ tmp104, a_global,
        @ lr needed for prologue  @
        str    r3, [r2, #0] @ tmp104, irq2_Variable
        ldmfd  sp!, {r2, r3}
        subs   pc, lr, #4
.L12:
        .align 2
.L11:
        .word  a_global
        .word  irq2_Variable
        .size  isr_2, .-isr_2
```

Man sieht deutlich, dass für diese InterruptServiceRoutine schon die erste Optimierungsstufe den Aufwand so verkürzt, dass die anderen Optimierungsstufen prinzipiell keinen Fortschritt bringen.

Allerdings gibt es einen kleinen Unterschied, abgesehen von der Position des Kommentars, zwischen der ersten und den anderen Optimierungsstufen in der vierten und fünften Code-Zeile (mit Funktionalität). Diese sind zwischen O1 und O2,3,Os nämlich vertauscht.

Interessant: **Die Optimierungsstufe 3 ersetzt zudem den Aufruf der Schleifen-Funktion in main() durch das Ergebnis der Schleife.**

Zunächst die Betrachtung der Auswirkung von volatile auf die Funktion test_global_schleife1()

ARM 7 tdmi, keine Optimierung

```
        .align 2
        .global      test_global_schleife1
        .type  test_global_schleife1, %function
test_global_schleife1:
        @ args = 0, pretend = 0, frame = 4
        @ frame_needed = 1, uses_anonymous_args = 0
        @ basic block 1
        mov    ip, sp @,
        stmfd  sp!, {fp, ip, lr, pc}     @,
        sub    fp, ip, #4   @,,
        sub    sp, sp, #4   @,,
        mov    r3, #0 @ tmp104,
        str    r3, [fp, #-16]      @ tmp104, i
        ldr    r2, .L6      @ tmp105,
        mov    r3, #0 @ tmp106,
        str    r3, [r2, #0] @ tmp106, a_global
        b      .L2   @
.L3:
        @ basic block 2
        ldr    r2, [fp, #-16]      @ i, i
        ldr    r3, [fp, #-16]      @ i, i
        mul    r2, r3, r2   @ D.1241, i, i
        ldr    r3, .L6      @ tmp109,
        ldr    r3, [r3, #0] @ a_global.0, a_global
        add    r2, r2, r3   @ D.1243, D.1241, a_global.0
        ldr    r3, .L6      @ tmp110,
        str    r2, [r3, #0] @ D.1243, a_global
        ldr    r3, [fp, #-16]      @ i, i
        add    r3, r3, #1   @ tmp112, i,
        str    r3, [fp, #-16]      @ tmp112, i
.L2:
        @ basic block 3
        ldr    r3, [fp, #-16]      @ i, i
        cmp    r3, #9 @ i,
        ble    .L3   @,
        @ basic block 7
        ldmfd  sp, {r3, fp, sp, pc}      @
.L7:
        .align 2
.L6:
        .word  a_global
```

ARM 7 tdmi, erste Optimierung

```
        .align 2
        .global      test_global_schleife1
        .type  test_global_schleife1, %function
test_global_schleife1:
        @ args = 0, pretend = 0, frame = 0
        @ frame_needed = 0, uses_anonymous_args = 0
        @ link register save eliminated.
        @ basic block 0
        @ lr needed for prologue  @
        mov    r2, #0 @ tmp107,
        ldr    r3, .L7      @ tmp106,
        str    r2, [r3, #0] @ tmp107, a_global
        mov    r1, r3 @ tmp108, tmp106
.L2:
        @ basic block 1
        ldr    r3, [r1, #0] @ a_global.0, a_global
        mla    r3, r2, r2, r3      @ D.1243, i, i, a_global.0
        str    r3, [r1, #0] @ D.1243, a_global
        add    r2, r2, #1   @ i, i,
        cmp    r2, #10      @ i,
        bne    .L2   @,
        @ basic block 2
        bx     lr    @
.L8:
        .align 2
.L7:
        .word  a_global
        .size  test_global_schleife1, .-test_global_schleife1
```

`.size test_global_schleife1, .-test_global_schleife1`	
zweite Optimierung	dritte Optimierung
`.align 2` `.global test_global_schleife1` `.type test_global_schleife1, %function` `test_global_schleife1:` `@ args = 0, pretend = 0, frame = 0` `@ frame_needed = 0, uses_anonymous_args = 0` `@ link register save eliminated.` `@ basic block 0` `ldr r0, .L8 @ tmp111,` `mov r3, #0 @ tmp107,` `str r3, [r0, #0] @ tmp107, a_global` `@ lr needed for prologue @` `mov r1, r3 @ i, tmp107` `.L2:` `@ basic block 1` `ldr r3, [r0, #0] @ a_global.0, a_global` `mla r2, r1, r1, r3 @ D.1243, i, i, a_global.0` `add r1, r1, #1 @ i, i,` `cmp r1, #10 @ i,` `str r2, [r0, #0] @ D.1243, a_global` `bne .L2 @,` `@ basic block 2` `bx lr @` `.L9:` `.align 2` `.L8:` `.word a_global` `.size test_global_schleife1, .-test_global_schleife1`	`.align 2` `.global test_global_schleife1` `.type test_global_schleife1, %function` `test_global_schleife1:` `@ args = 0, pretend = 0, frame = 0` `@ frame_needed = 0, uses_anonymous_args = 0` `@ link register save eliminated.` `@ basic block 0` `ldr r1, .L3 @ tmp120,` `mov r3, #0 @ tmp121,` `str r3, [r1, #0] @ tmp121, a_global` `ldr r2, [r1, #0] @ a_global.40, a_global` `str r2, [r1, #0] @ a_global.40, a_global` `ldr r3, [r1, #0] @ a_global.41, a_global` `add r3, r3, #1 @ temp.32, a_global.41,` `str r3, [r1, #0] @ temp.32, a_global` `ldr r2, [r1, #0] @ a_global.42, a_global` `add r2, r2, #4 @ temp.33, a_global.42,` `str r2, [r1, #0] @ temp.33, a_global` `ldr r3, [r1, #0] @ a_global.43, a_global` `add r3, r3, #9 @ temp.34, a_global.43,` `str r3, [r1, #0] @ temp.34, a_global` `ldr r2, [r1, #0] @ a_global.44, a_global` `add r2, r2, #16 @ temp.35, a_global.44,` `str r2, [r1, #0] @ temp.35, a_global` `ldr r3, [r1, #0] @ a_global.45, a_global` `add r3, r3, #25 @ temp.36, a_global.45,` `str r3, [r1, #0] @ temp.36, a_global` `ldr r2, [r1, #0] @ a_global.46, a_global` `add r2, r2, #36 @ temp.37, a_global.46,` `str r2, [r1, #0] @ temp.37, a_global` `ldr r3, [r1, #0] @ a_global.47, a_global` `add r3, r3, #49 @ temp.38, a_global.47,` `str r3, [r1, #0] @ temp.38, a_global` `ldr r2, [r1, #0] @ a_global.48, a_global` `add r2, r2, #64 @ temp.39, a_global.48,` `str r2, [r1, #0] @ temp.39, a_global` `ldr r3, [r1, #0] @ a_global.0, a_global` `add r3, r3, #81 @ D.1243, a_global.0,` `@ lr needed for prologue @` `str r3, [r1, #0] @ D.1243, a_global` `bx lr @`

	`.L4:` `.align 2` `.L3:` `.word a_global` `.size test_global_schleife1, .-test_global_schleife1`
Speicheroptimierung	
`test_global_schleife1:` `@ args = 0, pretend = 0, frame = 0` `@ frame_needed = 0, uses_anonymous_args = 0` `@ link register save eliminated.` `@ basic block 0` `ldr r3, .L7 @ tmp106,` `mov r2, #0 @ tmp107,` `str r2, [r3, #0] @ tmp107, a_global` `@ lr needed for prologue @` `mov r1, r2 @ i, tmp107` `.L2:` `@ basic block 1` `ldr r3, .L7 @ tmp108,` `ldr r2, [r3, #0] @ a_global.0, a_global` `mla r2, r1, r1, r2 @ D.1243, i, i, a_global.0` `add r1, r1, #1 @ i, i,` `cmp r1, #10 @ i,` `str r2, [r3, #0] @ D.1243, a_global` `bne .L2 @,` `@ basic block 2` `bx lr @` `.L8:` `.align 2` `.L7:` `.word a_global` `.size test_global_schleife1, .-test_global_schleife1`	Den Einfluss des Schlüsselwortes volatile sieht man deutlich. In allen Schleifen-Übersetzungen wird das Zwischenergebnis vor jedem Sprung in die globale Variable gespeichert. Bei der Optimierungsstufe 3 wird dieses besonders deutlich, denn hier wird die Schleife komplett entrollt. Die Quadrierung des Schleifenindexes und die Addition dieses Quadrates zur globalen Variable wird nun verwirklicht, indem dem Schleifenzählerquadrat entsprechende konstante Werte addiert werden. Auch hier wird jedes Zwischenergebnis direkt zurück geschrieben. ***Interessant:*** **Die Optimierungsstufe 3 ersetzt auch hier den Aufruf der Schleifen-Funktion in main() durch das Ergebnis der Schleife.**

Nun die Betrachtung der Auswirkung von volatile auf die Interruptserviceroutine isr_2()

ARM 7 tdmi, keine Optimierung

```
      .align 2
      .global      isr_2
      .type  isr_2, %function
isr_2:
      @ Interrupt Service Routine.
      @ args = 0, pretend = 0, frame = 0
      @ frame_needed = 1, uses_anonymous_args = 0
      @ basic block 1
      str    ip, [sp, #-4]!      @,
      mov    ip, sp @,
      stmfd  sp!, {r2, r3, fp, ip, lr, pc}    @,
      sub    fp, ip, #4   @,,
      ldr    r3, .L10     @ tmp104,
      ldr    r2, [r3, #0] @ a_global.1, a_global
      ldr    r3, .L10+4   @ tmp105,
      str    r2, [r3, #0] @ a_global.1, irq2_Variable
      ldr    r3, .L10+4   @ tmp106,
      ldr    r3, [r3, #0] @ irq2_Variable.2, irq2_Variable
      add    r2, r3, #1   @ D.1249, irq2_Variable.2,
      ldr    r3, .L10+4   @ tmp107,
      str    r2, [r3, #0] @ D.1249, irq2_Variable
      sub    sp, fp, #20
      ldmfd  sp, {r2, r3, fp, sp, lr}
      ldmfd  sp!, {ip}
      subs   pc, lr, #4
.L11:
      .align 2
.L10:
      .word  a_global
      .word  irq2_Variable
      .size  isr_2, .-isr_2
```

ARM 7 tdmi, erste Optimierung

```
      .align 2
      .global      isr_2
      .type  isr_2, %function
isr_2:
      @ Interrupt Service Routine.
      @ args = 0, pretend = 0, frame = 0
      @ frame_needed = 0, uses_anonymous_args = 0
      @ link register save eliminated.
      @ basic block 0
      stmfd  sp!, {r2, r3}@,
      @ lr needed for prologue  @
      ldr    r3, .L11     @ tmp102,
      ldr    r3, [r3, #0] @ a_global.1, a_global
      add    r3, r3, #1   @ tmp104, a_global.1,
      ldr    r2, .L11+4   @ tmp103,
      str    r3, [r2, #0] @ tmp104, irq2_Variable
      ldmfd  sp!, {r2, r3}
      subs   pc, lr, #4
.L12:
      .align 2
.L11:
      .word  a_global
      .word  irq2_Variable
      .size  isr_2, .-isr_2
```

zweite Optimierung

```
      .align 2
      .global      isr_2
      .type  isr_2, %function
isr_2:
      @ Interrupt Service Routine.
      @ args = 0, pretend = 0, frame = 0
      @ frame_needed = 0, uses_anonymous_args = 0
      @ link register save eliminated.
      @ basic block 0
```

dritte Optimierung

```
      .align 2
      .global      isr_2
      .type  isr_2, %function
isr_2:
      @ Interrupt Service Routine.
      @ args = 0, pretend = 0, frame = 0
      @ frame_needed = 0, uses_anonymous_args = 0
      @ link register save eliminated.
      @ basic block 0
```

`stmfd sp!, {r2, r3}@,` `ldr r3, .L12 @ tmp102,` `ldr r2, [r3, #0] @ a_global.1, a_global` `ldr r3, .L12+4 @ tmp103,` `add r2, r2, #1 @ tmp104, a_global.1,` `@ lr needed for prologue @` `str r2, [r3, #0] @ tmp104, irq2_Variable` `ldmfd sp!, {r2, r3}` `subs pc, lr, #4` `.L13:` `.align 2` `.L12:` `.word a_global` `.word irq2_Variable` `.size isr_2, .-isr_2`	`stmfd sp!, {r2, r3}@,` `ldr r3, .L7 @ tmp102,` `ldr r2, [r3, #0] @ a_global.1, a_global` `ldr r3, .L7+4 @ tmp103,` `add r2, r2, #1 @ tmp104, a_global.1,` `@ lr needed for prologue @` `str r2, [r3, #0] @ tmp104, irq2_Variable` `ldmfd sp!, {r2, r3}` `subs pc, lr, #4` `.L8:` `.align 2` `.L7:` `.word a_global` `.word irq2_Variable` `.size isr_2, .-isr_2`
Speicheroptimierung	
`.align 2` `.global isr_2` `.type isr_2, %function` `isr_2:` `@ Interrupt Service Routine.` `@ args = 0, pretend = 0, frame = 0` `@ frame_needed = 0, uses_anonymous_args = 0` `@ link register save eliminated.` `@ basic block 0` `stmfd sp!, {r2, r3}@,` `ldr r3, .L11 @ tmp102,` `ldr r3, [r3, #0] @ a_global.1, a_global` `ldr r2, .L11+4 @ tmp103,` `add r3, r3, #1 @ tmp104, a_global.1,` `@ lr needed for prologue @` `str r3, [r2, #0] @ tmp104, irq2_Variable` `ldmfd sp!, {r2, r3}` `subs pc, lr, #4` `.L12:` `.align 2` `.L11:` `.word a_global` `.word irq2_Variable` `.size isr_2, .-isr_2`	Das Einfluss des Schlüsselwortes volatile angewandt auf die globale Variable a_global hat im Code der ISR keine Auswirkung.

Es sei verdeutlicht, dass in der ISR nur lesend auf die globale Variable a_global zugegriffen wurde.
Daher nun eine leichte Modifikation der isr, so dass auf die globale Variable a_global auch schreibend zugegriffen wird.

C-Code:

```
int irq2_Variable; /* Globale Variable für die InterruptServiceRoutine 2*/
volatile int a_global; /* wird sowohl durch Interruptserviceroutine als auch durch die Schleife verändert*/

/*****************************************************************************/
/** Zählschleife */
void test_global_schleife1()
{/*Modifiziert die globale Integer-Variable a_global*/
    int i;
    for(i = 0, a_global = 0; i < 10; i++ )
    {
      a_global += i*i;
    }
}


/** Interrupt */
void __attribute__((interrupt("IRQ"))) isr_2(void)
{
      irq2_Variable=a_global;
      a_global = 399;
}


/*****************************************************************************/

int main()
{
      test_global_schleife1();
      return 0;
}
/*****************************************************************************/
```

Der Assembler-Code für die Funktion test_global_schleife1( ) hat sich im Vergleich zur Version, in der die ISR nur lesend auf a_global zugreift, nicht verändert. Dies bedeutet, dass ein Interrupt einen **„geschlossenen Block“** unterbrechen darf.

Als Beispiel eines geschlossenen Blocks hier ein Pseudo-Code:

```
Load R3, Adresse;  // Inhalt bei Adresse wird in R3 geladen
Inc R3;            // Inhalt von R3 wird einmal inkrementiert
Store R3, Adresse; // Inhalt von R3 wird bei Adresse abgespeichert.
```

hier der entsprechende ARM-Code (Optimierungsstufe 3):

```
ldr    r3, [r1, #0] @ a_global.41, a_global
add    r3, r3, #1   @ temp.32, a_global.41,
str    r3, [r1, #0] @ temp.32, a_global
```

Hier die Betrachtung der Auswirkung von volatile auf die Interruptserviceroutine isr_2() mit Schreibzugriff auf a_global

ARM 7 tdmi, keine Optimierung

```
      .align 2
      .global      isr_2
      .type  isr_2, %function
isr_2:
      @ Interrupt Service Routine.
      @ args = 0, pretend = 0, frame = 0
      @ frame_needed = 1, uses_anonymous_args = 0
      @ basic block 1
      str    ip, [sp, #-4]!      @,
      mov    ip, sp @,
      stmfd  sp!, {r2, r3, fp, ip, lr, pc}    @,
      sub    fp, ip, #4   @,,
      ldr    r3, .L10     @ tmp102,
      ldr    r2, [r3, #0] @ a_global.1, a_global
      ldr    r3, .L10+4   @ tmp103,
      str    r2, [r3, #0] @ a_global.1, irq2_Variable
      ldr    r2, .L10     @ tmp104,
      mov    r3, #396     @ tmp105,
      add    r3, r3, #3   @ tmp105, tmp105,
      str    r3, [r2, #0] @ tmp105, a_global
      sub    sp, fp, #20
      ldmfd  sp, {r2, r3, fp, sp, lr}
      ldmfd  sp!, {ip}
      subs   pc, lr, #4
.L11:
      .align 2
.L10:
      .word  a_global
      .word  irq2_Variable
      .size  isr_2, .-isr_2
```

ARM 7 tdmi, erste Optimierung

```
      .align 2
      .global      isr_2
      .type  isr_2, %function
isr_2:
      @ Interrupt Service Routine.
      @ args = 0, pretend = 0, frame = 0
      @ frame_needed = 0, uses_anonymous_args = 0
      @ link register save eliminated.
      @ basic block 0
      stmfd  sp!, {r1, r2, r3}   @,
      @ lr needed for prologue   @
      ldr    r1, .L11     @ tmp102,
      ldr    r2, [r1, #0] @ a_global.1, a_global
      ldr    r3, .L11+4   @ tmp103,
      str    r2, [r3, #0] @ a_global.1, irq2_Variable
      mov    r3, #396     @ tmp106,
      add    r3, r3, #3   @ tmp105, tmp106,
      str    r3, [r1, #0] @ tmp105, a_global
      ldmfd  sp!, {r1, r2, r3}
      subs   pc, lr, #4
.L12:
      .align 2
.L11:
      .word  a_global
      .word  irq2_Variable
      .size  isr_2, .-isr_2
```

zweite Optimierung

```
      .align 2
      .global      isr_2
      .type  isr_2, %function
isr_2:
      @ Interrupt Service Routine.
      @ args = 0, pretend = 0, frame = 0
      @ frame_needed = 0, uses_anonymous_args = 0
      @ link register save eliminated.
      @ basic block 0
      stmfd  sp!, {r0, r1, r2, r3}     @,
```

dritte Optimierung

```
      .align 2
      .global      isr_2
      .type  isr_2, %function
isr_2:
      @ Interrupt Service Routine.
      @ args = 0, pretend = 0, frame = 0
      @ frame_needed = 0, uses_anonymous_args = 0
      @ link register save eliminated.
      @ basic block 0
      stmfd  sp!, {r0, r1, r2, r3}     @,
```

```
      ldr    r0, .L12     @ tmp102,
      ldr    r2, .L12+4   @ tmp103,
      ldr    r1, [r0, #0] @ a_global.1, a_global
      mov    r3, #396     @ tmp106,
      add    r3, r3, #3   @ tmp105, tmp106,
      @ lr needed for prologue  @
      str    r1, [r2, #0] @ a_global.1, irq2_Variable
      str    r3, [r0, #0] @ tmp105, a_global
      ldmfd  sp!, {r0, r1, r2, r3}
      subs   pc, lr, #4
.L13:
      .align 2
.L12:
      .word  a_global
      .word  irq2_Variable
      .size  isr_2, .-isr_2
```

```
      ldr    r0, .L7      @ tmp102,
      ldr    r2, .L7+4    @ tmp103,
      ldr    r1, [r0, #0] @ a_global.1, a_global
      mov    r3, #396     @ tmp106,
      add    r3, r3, #3   @ tmp105, tmp106,
      @ lr needed for prologue  @
      str    r1, [r2, #0] @ a_global.1, irq2_Variable
      str    r3, [r0, #0] @ tmp105, a_global
      ldmfd  sp!, {r0, r1, r2, r3}
      subs   pc, lr, #4
.L8:
      .align 2
.L7:
      .word  a_global
      .word  irq2_Variable
      .size  isr_2, .-isr_2
```

Speicheroptimierung

```
      .align 2
      .global      isr_2
      .type  isr_2, %function
isr_2:
      @ Interrupt Service Routine.
      @ args = 0, pretend = 0, frame = 0
      @ frame_needed = 0, uses_anonymous_args = 0
      @ link register save eliminated.
      @ basic block 0
      stmfd  sp!, {r1, r2, r3}   @,
      ldr    r1, .L11     @ tmp102,
      ldr    r3, .L11+4   @ tmp103,
      ldr    r2, [r1, #0] @ a_global.1, a_global
      str    r2, [r3, #0] @ a_global.1, irq2_Variable
      ldr    r3, .L11+8   @ tmp105,
      @ lr needed for prologue  @
      str    r3, [r1, #0] @ tmp105, a_global
      ldmfd  sp!, {r1, r2, r3}
      subs   pc, lr, #4
.L12:
      .align 2
.L11:
      .word  a_global
      .word  irq2_Variable
      .word  399
      .size  isr_2, .-isr_2
```

Das Einfluss des Schlüsselwortes volatile angewandt auf die globale Variable a_global hat im Code der ISR keine Auswirkung, obwohl hier nun schreibender Zugriff existiert.

Interessant: Bei der Speicheroptimierung, wird der konstante Wert „396“ nicht mit einem mov an a_global übergeben, sondern mit einem ldr.

Ein Ergebnis dieser Betrachtung: Da die Programmierung von Interrupt Service Routinen nicht im C-Standard beachtet wird, liegt es im Ermessen von Herstellern entsprechende Möglichkeiten ihrem Compiler mitzugeben, bzw es liegt beim Chip-Hersteller ob er C-Bibliotheken mitliefert, die es ermöglichen ISR auf Hochsprach-Ebene zu programmieren.

Der Programmierer muss dem Compiler mit dem Schlüsselwort „volatile“ mitteilen, welche Variablen durch andere Ereignisse (als Ereignisse von normalen Funktionen ausgeführt) verändert werden können.
Ebenso muss der Programmierer für das Problem der unterbrechbaren geschlossenen Blocks, die eigentlich nicht unterbrochen werden dürfen, eine Lösung finden; diese Lösung lässt sich wahrscheinlich nicht ohne Erweiterung der Sprachelemente auf der C-Sprachebene erreichen.

Schön für den syntaktischen Umgang eines Interrupts auf C-Syntax-Ebene wäre:

```
volatile int changable;


int bla(int input)
{
	int output;

	... interruptible code ...

	atomic_block  // code in this block is not interruptible
	{
		... something ...

	       changable = ... new value ...

	       ... something ...
	}

	... interruptible code ...

	return output;
}

void interrupt(breakable) Name_of_ISR(void)
{ // An other interrupt can start and break this code.
  ...
}

void interrupt(not_breakable) Name_of_other_ISR(void)
{ // No other interrupt can start
  ...
}
```