```
/* bmw_small-tinyasm.S */
/*
    This file is part of the AVR-Crypto-Lib.
    Copyright (C) 2009  Daniel Otte (daniel.otte@rub.de)

    This program is free software: you can redistribute it
 and/or modify
    it under the terms of the GNU General Public License a
s published by
    the Free Software Foundation, either version 3 of the
License, or
    (at your option) any later version.

    This program is distributed in the hope that it will b
e useful,
    but WITHOUT ANY WARRANTY; without even the implied war
ranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Pub
lic License
    along with this program.  If not, see <http://www.gnu.
org/licenses/>.
*/

/*
 * File:        bmw_small-tinyasm.S
 * Author:      Daniel Otte
 * Date:        2010-03-28
 * License:     GPLv3 or later
 * Description: implementation of BlueMidnightWish
 *
 */

#include "avr-asm-macros.S"

/*********************************************************
********************/
/*
  param a: r22:r23:r24:r25
  param s: r20
*/
shiftleft32:
        clr r0
        cpi r20, 8
        brlo bitrotateleft_1
        mov r25, r24
        mov r24, r23
        mov r23, r22
        clr r22
        subi r20, 8
        rjmp shiftleft32

/*********************************************************
********************/
/*
  param a: r22:r23:r24:r25
  param s: r20
*/
shiftright32:
        cpi r20, 8
        brlo bitshiftright
        mov r22, r23
        mov r23, r24
        mov r24, r25
        clr r25
        subi r20, 8
        rjmp shiftright32
bitshiftright:
        tst r20
        breq 20f
10:     lsr r25
        ror r24
        ror r23
        ror r22
        dec r20
        brne 10b
20: ret

/*********************************************************
********************/
/*
  param a: r22:r23:r24:r25
  param s: r20
*/
rotateleft32:
        cpi r20, 8
        brlo bitrotateleft
        mov r0, r25
        mov r25, r24
        mov r24, r23
        mov r23, r22
        mov r22, r0
        subi r20, 8
        rjmp rotateleft32
bitrotateleft:
    mov r0, r25
bitrotateleft_1:
        tst r20
        breq 20f
10:
        lsl r0
        rol r22
        rol r23
        rol r24
        rol r25
        dec r20
        brne 10b
20: ret


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

s_table:
s0:  .byte 1, 3, 4,19
s1:  .byte 1, 2, 8,23
s2:  .byte 2, 1,12,25
s3:  .byte 2, 2,15,29
s4:  .byte 1, 0, 0, 0
s5:  .byte 2, 0, 0, 0

eor_r22_in_r16:
        eor r16, r22
        eor r17, r23
        eor r18, r24
        eor r19, r25
        ret

/*
  param x: r22:r23:r24:25
  param s: r20
*/
sn:
        push_range 2, 5
        push r17
        push r19
        ldi r30, lo8(s_table)
        ldi r31, hi8(s_table)
        lsl r20
        lsl r20
        add r30, r20
        adc r31, r1
        movw r2, r22
        movw r4, r24
        lpm r20, Z+
        rcall shiftright32
        movw r16, r22
        movw r18, r24
;---
        movw r22, r2
        movw r24, r4
        lpm r20, Z+
        rcall shiftleft32
        rcall eor_r22_in_r16
;---
        movw r22, r2
        movw r24, r4
        lpm r20, Z+
        rcall rotateleft32
        rcall eor_r22_in_r16
;---
        movw r22, r2
        movw r24, r4
        lpm r20, Z+
        rcall rotateleft32
        eor r22, r16
        eor r23, r17
        eor r24, r18
        eor r25, r19
        pop r19
        pop r17
        pop_range 2, 5
        ret

/*********************************************************
********************/
/*
  param dest: r26:r27 (X)
  param src:  r30:r31 (Z)
  param len:  r20
*/
memxor_short:
;       tst r20
;       breq memxor_exit
10: ld r21, X
        ld r22, Z+
        eor r21, r22
        st X+, r21
        dec r20
        brne 10b
memxor_exit:
        ret

/*********************************************************
********************/
q0 = 2
q1 = 3
h0 = 4
```

```
h1 = 5
m0 = 6
m1 = 7

add_hx_to_w:
        movw r26, h0
        add r26, r16
        adc r27, r1
        ld r22, Y
        ldd r23, Y+1
        ldd r24, Y+2
        ldd r25, Y+3
        lsl r20
        rol r21
        brcs 30f
        /* addition */
        ld r0, X+
        add r22, r0
        ld r0, X+
        adc r23, r0
        ld r0, X+
        adc r24, r0
        ld r0, X+
        adc r25, r0
        rjmp 50f
30: /* substract */
        ld r0, X+
        sub r22, r0
        ld r0, X+
        sbc r23, r0
        ld r0, X+
        sbc r24, r0
        ld r0, X+
        sbc r25, r0
50:
        st Y+, r22
        st Y+, r23
        st Y+, r24
        st Y+, r25
        ret

/**********************************************************
********************/
load32_from_X:
        ld r22, X+
        ld r23, X+
        ld r24, X+
        ld r25, X+
        ret

load32_from_Y:
        ld r22, Y+
        ld r23, Y+
        ld r24, Y+
        ld r25, Y+
        ret

add_X_to_32:
        ld r0, X+
        add r22, r0
        ld r0, X+
        adc r23, r0
        ld r0, X+
        adc r24, r0
        ld r0, X+
        adc r25, r0
        ret

/**********************************************************
********************/
/*
  param q:  r28:r29 (Y)
  param h:  r26:r27 (X)
  param m:  r30:r31 (Z)
*/

f0_hacktable:
        .byte 0x03, 0x11
        .byte 0xDD, 0xB3
        .byte 0x2A, 0x79
        .byte 0x07, 0xAA
        .byte 0x51, 0xC2
f0_indextable:
        .byte 5*4,7*4,10*4,13*4,14*4
;       .byte 0 ; just for alignment
f0_s_table:
        .byte 0,1,2,3,4
        .byte 0,1,2,3,4
        .byte 0,1,2,3,4
;       .byte 0

f0:
        movw h0, r26
        movw q0, r28
        movw m0, r30
;--- DBG
;       push_range 22, 25
;       movw r24, r26
;       ldi r22, 'H'
;       rcall printX
;       pop_range  22, 25
;--- END DBG
;--- DBG
;       push_range 22, 25
;       movw r24, r30
;       ldi r22, 'M'
;       rcall printX
;       pop_range  22, 25
;--- END DBG
        /* xor m into h */
        ldi r20, 64
        rcall memxor_short
        movw r30, m0
        movw r26, h0

        /* set q to zero */
        ldi r22, 64
10:     st Y+, r1
        dec r22
        brne 10b
        movw r28, q0
        /* calculate W and store it in Q */
        ldi r19, 5
30:
        ldi r18, 16
        /* load initial index */
        ldi r30, lo8(f0_indextable-1)
        ldi r31, hi8(f0_indextable-1)
        add r30, r19
        adc r31, r1
        lpm r16, Z
        /* load values from hacktable */
        ldi r30, lo8(f0_hacktable-2)
        ldi r31, hi8(f0_hacktable-2)
        lsl r19
        add r30, r19
        adc r31, r1
        lsr r19
        lpm r21, Z+
        lpm r20, Z
40:
        call add_hx_to_w
        subi r16, -4
        andi r16, 0x0f<<2
        dec r18
        brne 40b
        movw r28, q0
        dec r19
        brne 30b
        movw r26, h0
;--- DBG
;       push_range 22, 25
;       movw r24, r28
;       ldi r22, 'W'
;       rcall printX
;       pop_range  22, 25
;--- END DBG
        /* xor m into h */
        ldi r20, 64
        movw r26, h0
        movw r30, m0
        rcall memxor_short
        sbiw r26, 60
;---
        ldi r30, lo8(f0_s_table)
        ldi r31, hi8(f0_s_table)
        ldi r21, 15
        mov r8, r21
50:
        ldd r22, Y+0
        ldd r23, Y+1
        ldd r24, Y+2
        ldd r25, Y+3
        lpm r20, Z+
        movw r2, r30
        rcall sn
        movw r30, r2

        rcall add_X_to_32

        st Y+, r22
        st Y+, r23
        st Y+, r24
        st Y+, r25
        dec r8
        brne 50b
;---
        ldd r22, Y+0
        ldd r23, Y+1
        ldd r24, Y+2
        ldd r25, Y+3
        clr r20
        rcall sn
        movw r30, r2
        movw r26, h0
        rcall add_X_to_32
        sbiw r26, 4
        std Y+0, r22
        std Y+1, r23
        std Y+2, r24
        std Y+3, r25
        sbiw r28, 15*4
```

```
        movw r20, h0
        movw r22, m0
        ret

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

const_lut:
        .long 0x55555550, 0x5aaaaaa5, 0x5ffffffa, 0x655555
4f
        .long 0x6aaaaaa4, 0x6ffffff9, 0x7555554e, 0x7aaaaa
a3
        .long 0x7ffffff8, 0x8555554d, 0x8aaaaaa2, 0x8fffff
f7
        .long 0x9555554c, 0x9aaaaaa1, 0x9ffffff6, 0xa55555
4b

/*******************************************************
*********************
* uint32_t addelment(uint8_t j, const uint32_t* m, const u
int32_t* h){
*       uint32_t r;
*       r  = pgm_read_dword(k_lut+j);
*       r += rotl_addel(((uint32_t*)m)[j&0xf], j+0);
*       r += rotl_addel(((uint32_t*)m)[(j+3)&0xf], j+3);
*       r -= rotl_addel(((uint32_t*)m)[(j+10)&0xf], j+10);
*       r ^= ((uint32_t*)h)[(j+7)&0xf];
*       return r;
* }
* param j: r24
* param m: r22:r23
* param h: r20:r21
*/
j    = 16
acc2 =  8
acc3 =  9
h0   = 10
h1   = 11
m0   = 12
m1   = 13
acc0 = 14
acc1 = 15

add32_to_acc:
        add acc0, r22
        adc acc1, r23
        adc acc2, r24
        adc acc3, r25
        ret

eor32_to_acc:
        eor acc0, r22
        eor acc1, r23
        eor acc2, r24
        eor acc3, r25
        ret

load_acc_from_X:
        ld acc0, X+
        ld acc1, X+
        ld acc2, X+
        ld acc3, X+
        ret

add_acc_to_Z:
        ld r0, Z
        add r0, acc0
        st Z+, r0
        ld r0, Z
        adc r0, acc1
        st Z+, r0
        ld r0, Z
        adc r0, acc2
        st Z+, r0
        ld r0, Z
        adc r0, acc3
        st Z+, r0
        ret

load_rotate_add_M:
        andi r20, 0x0f
        mov r0, r20
        lsl r0
        lsl r0
        movw r26, m0
        add r26, r0
        adc r27, r1
        ld r22, X+
        ld r23, X+
        ld r24, X+
        ld r25, X+
        inc r20
        rcall rotateleft32
        brts 10f
        rcall add32_to_acc
        ret
10:     sub acc0, r22
        sbc acc1, r23
        sbc acc2, r24
        sbc acc3, r25
        ret

addelement:
        mov j, r24
        movw h0, r20
        movw m0, r22
        lsl r24
        lsl r24
        mov r28, r24
        ldi r30, lo8(const_lut)
        ldi r31, hi8(const_lut)
        add r30, r24
        adc r31, r1
        lpm acc0, Z+
        lpm acc1, Z+
        lpm acc2, Z+
        lpm acc3, Z+
        clt
        mov r20, j
        rcall load_rotate_add_M
        mov r20, j
        subi r20, -3
        rcall load_rotate_add_M
        mov r20, j
        set
        subi r20, -10
        rcall load_rotate_add_M
        lsl j
        lsl j
        subi j, -7*4
        andi j, 0x3f
        movw r26, h0
        add r26, j
        adc r27, r1
        ld r0, X+
        eor acc0, r0
        ld r0, X+
        eor acc1, r0
        ld r0, X+
        eor acc2, r0
        ld r0, X+
        eor acc3, r0
;---
        ret

/*******************************************************
********************/
/*
  param q: r26:r27
  param m: r22:r23
  param h: r20:r21
  param j: r24
*/

expand_intro:
        push_range 20, 27
;       push r24
        rcall addelement
;       pop r24
        pop_range 20, 27
        lsl r24
        lsl r24
        add r26, r24
        adc r27, r1
        ret
expand1:
        rcall expand_intro
        ldi r19, 1
10:
        rcall load32_from_X
        mov r20, r19
        andi r20, 3
        rcall sn
        rcall add32_to_acc
        inc r19
        cpi r19, 17
        brne 10b
expand1_exit:
;       adiw r26, 63
        st X+, acc0
        st X+, acc1
        st X+, acc2
        st X+, acc3
        ret

/*******************************************************
********************/
/*
  param q: r26:r27
  param m: r22:r23
  param h: r20:r21
  param j: r24
*/

expand2_rot_table:
        .byte 0,3,0,7,0,13,0,16,0,19,0,23,0,27

expand2:
        rcall expand_intro
        ldi r19, 14
        ldi r30, lo8(expand2_rot_table)
```

```
        ldi r31, hi8(expand2_rot_table)
10:
        rcall load32_from_X
        mov r20, r19
        lpm r20, Z+
        rcall rotateleft32
        rcall add32_to_acc
        dec r19
        brne 10b
        rcall load32_from_X
        ldi r20, 4
        rcall sn
        rcall add32_to_acc
        rcall load32_from_X
        ldi r20, 5
        rcall sn
        rcall add32_to_acc

        rjmp expand1_exit

/**********************************************************
********************/
/*
  param q: r24:r25
  param m: r22:r23
  param h: r20:r21
*/
/* for calling expand1/2
  param q: r26:r27
  param m: r22:r23
  param h: r20:r21
  param j: r24
*/
f1:
        movw r2, r24
        movw r4, r22
        movw r6, r20
        movw r26, r2
;       movw r22, r4
;   movw r20, r6
        clr r24
        rcall expand1
        movw r26, r2
        movw r22, r4
    movw r20, r6
        ldi r24, 1
        rcall expand1
        ldi r17, 2
10:     movw r26, r2
        movw r22, r4
        movw r20, r6
        mov r24, r17
        rcall expand2
        inc r17
        sbrs r17, 4
        rjmp 10b
        movw r24, r2
        movw r22, r4
        movw r20, r6
        ret

/**********************************************************
********************/
/*
  param q: r24:r25
  param m: r22:r23
  param h: r20:r21
*/
f2_1_shift_table:
        .byte 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x
00
        .byte 0x2B, 0x64, 0x66, 0x03, 0x51, 0x55, 0x87, 0x
55
f2_2_shift_table:
        .byte (2<<1), (7<<1), (4<<1), (3<<1), (4<<1)+1, (6
<<1)+1, (6<<1)
        .byte 0 ; just for alignment
acc2  =  8
acc3  =  9
acc0  = 14
acc1  = 15
xl0   =  2
xl1   =  3
xl2   =  4
xl3   =  5
xh0   =  6
xh1   =  7
xh2   = 10
xh3   = 11
q16_0 = 12
q16_1 = 13
h0    =  18
h1    =  19
f2:
        movw r26, r24
        /* calc XL */
        adiw r26, 63
        adiw r26,  1
        movw q16_0, r26
        clr xl0
        clr xl1
        clr xl2
        clr xl3
        ldi r17, 8
10:     ld r0, X+
        eor xl0, r0
        ld r0, X+
        eor xl1, r0
        ld r0, X+
        eor xl2, r0
        ld r0, X+
        eor xl3, r0
        dec r17
        brne 10b
;--- /* calc XH */
        movw xh0, xl0
        movw xh2, xl2
        ldi r17, 8
10:     ld r0, X+
        eor xh0, r0
        ld r0, X+
        eor xh1, r0
        ld r0, X+
        eor xh2, r0
        ld r0, X+
        eor xh3, r0
        dec r17
        brne 10b
;--- DBG
;       push_range 22, 25
;       movw r22, xl0
;       movw r24, xl2
;       rcall print32
;       movw r22, xh0
;       movw r24, xh2
;       rcall print32
;       pop_range 22, 25
;--- END DBG

;--- /* calc first half of h0..h15 */
        movw h0, r20
        movw r28, r22
        movw r26, q16_0
        ldi r17, 16
10:
        ld acc0, Y+
        ld acc1, Y+
        ld acc2, Y+
        ld acc3, Y+
;---
        ldi r30, lo8(f2_1_shift_table-1)
        ldi r31, hi8(f2_1_shift_table-1)
        movw r22, xh0
        movw r24, xh2
        add r30, r17
        adc r31, r1
        lpm r20, Z
        mov r1, r20
        andi r20, 0x0f
        clt
        cpi r17, 16
        breq 20f
        cpi r17, 11
        brne 21f
20:     set
21:     brts 25f
        rcall shiftright32
        rjmp 26f
25:     rcall shiftleft32
26: rcall eor32_to_acc
;---
        rcall load32_from_X
        mov r20, r1
        clr r1
        swap r20
        andi r20, 0x0f
        brts 27f
        rcall shiftleft32
        rjmp 28f
27:     rcall shiftright32
28:     rcall eor32_to_acc
;---
        movw r30, h0
        st Z+, acc0
        st Z+, acc1
        st Z+, acc2
        st Z+, acc3
        movw h0, r30
;---
        dec r17
        brne 10b
;-----
        sbiw r26, 4*8 /* X points to q[24] */
        movw r28, r26
        sbiw r28, 63
        sbiw r28, 33 /* Y points to q[0] */
        sbiw r30, 63
        sbiw r30,  1 /* Z points to h0 */
        ldi r17, 8
10:     movw acc0, xl0
        movw acc2, xl2
```

```
	rcall load32_from_X
	rcall eor32_to_acc
	rcall load32_from_Y
	rcall eor32_to_acc
	rcall add_acc_to_Z
	dec r17
	brne 10b
	sbiw r26, 9*4 /* X points to q[23] */
	rcall load_acc_from_X
	eor acc1, xl0
	eor acc2, xl1
	eor acc3, xl2
	rcall load32_from_Y
	rcall eor32_to_acc
	rcall add_acc_to_Z
;---
	sbiw r26, 8*4 /* X points to q[16] */
	mov h0, r30
	ldi r17, 7
10:
	ldi r30, lo8(f2_2_shift_table-1)
	ldi r31, hi8(f2_2_shift_table-1)
	add r30, r17
	adc r31, r1
	lpm r20, Z
	rcall load_acc_from_X
	movw r22, xl0
	movw r24, xl2
	lsr r20
	brcc 20f
	rcall shiftleft32
	rjmp 21f
20:	rcall shiftright32
21:
	rcall eor32_to_acc
	rcall load32_from_Y
	rcall eor32_to_acc
	movw r30, h0
	rcall add_acc_to_Z
	movw h0, r30
	dec r17
	brne 10b
;-----
	sbiw r30, 8*4 /* Z points to h8 */
	movw r26, r30
	sbiw r26, 4*4 /* X points to h4 */
	ldi r17, 8
	ldi r18, 9
10:
	rcall load32_from_X
	mov r20, r18
	rcall rotateleft32
	movw acc0, r22
	movw acc2, r24
	rcall add_acc_to_Z
	inc r18
	cpi r17, 5
	breq 20f
	dec r17
	brne 10b
	ret
20: sbiw r26, 8*4
	dec r17
	rjmp 10b

/**********************************************************
*********************/
/*
  param ctx:  r24:r25
  param msg:  r22:r23
*/
/* f0
  param q:  r28:r29 (Y)
  param h:  r26:r27 (X)
  param m:  r30:r31 (Z)
*/
/* f1
  param q: r24:r25
  param m: r22:r23
  param h: r20:r21
*/
/* f2
  param q: r24:r25
  param m: r22:r23
  param h: r20:r21
*/
.global bmw_small_nextBlock
.global bmw224_nextBlock
.global bmw256_nextBlock
bmw_small_nextBlock:
bmw224_nextBlock:
bmw256_nextBlock:
	push_range 28, 29
	push_range  2, 17
	stack_alloc_large 32*4, r28, r29
	adiw r28, 1
;	push_range 28, 29 /* push Q */
;	push_range 22, 25 /* push M & H */
	/* increment counter */
	movw r26, r24
	movw r2, r26
	adiw r26, 63
	adiw r26,  1
	rcall load_acc_from_X
	ldi r19, 1
	add acc0, r19
	adc acc1, r1
	adc acc2, r1
	adc acc3, r1
	st -X, acc3
	st -X, acc2
	st -X, acc1
	st -X, acc0
	/* call f0 */
	movw r30, r22
	movw r26, r24
	rcall f0
	/* call f1*/
	movw r24, r28

;	rcall printQ
	rcall f1
	/* call f2 */
;	pop_range 20, 25
;	push_range 20, 25
;	rcall printQ
;	push r20
;	push r21
	call f2
;--- DBG
;	pop r25
;	pop r24
;	ldi r22, 'H'
;	rcall printX
;--- END DBG
	stack_free_large3 32*4
	pop_range  2, 17
	pop_range 28, 29
	ret

/**********************************************************
*********************/
/*
  param ctx:  r24:r25
  param msg:  r22:r23
  param len:  r20:r21
*/
ctx0 =  2
ctx1 =  3
blc0 =  4
blc1 =  5
len0 = 28
len1 = 29
buf0 =  6
buf1 =  7

.global bmw_small_lastBlock
.global bmw224_lastBlock
.global bmw256_lastBlock
bmw_small_lastBlock:
bmw224_lastBlock:
bmw256_lastBlock:
/*	while(length_b >= BMW_SMALL_BLOCKSIZE){
		bmw_small_nextBlock(ctx, block);
		length_b -= BMW_SMALL_BLOCKSIZE;
		block = (uint8_t*)block + BMW_SMALL_BLOCKS
IZE_B;
	}
*/
	push_range 2, 7
	push_range 28, 29
	movw ctx0, r24
	movw blc0, r22
	movw len0, r20
1:
	cpi len1, hi8(512)
	brlo 2f
	movw r24, ctx0
	movw r22, blc0
	rcall bmw_small_nextBlock
	ldi r24, 64
	add blc0, r24
	adc blc1, r1
	subi len1, hi8(512)
	rjmp 1b
2:
/*	struct {
		uint8_t  buffer[64];
		uint32_t ctr;
	} pctx;
*/
	stack_alloc_large 68
	adiw r30, 1
	movw buf0, r30
/*	memset(pctx.buffer, 0, 64);
	memcpy(pctx.buffer, block, (length_b+7)/8);
	pctx.buffer[length_b>>3] |= 0x80 >> (length_b&0x07
);
*/	movw r24, len0
	lsr r25
	ror r24
```

```
        lsr r24
        lsr r24
        ldi r23, 63
        sub r23, r24
        movw r26, blc0
        tst r24
        breq 301f
        /* copy (#r24) bytes to stack buffer */
30: ld r20, X+
        st Z+, r20
        dec r24
        brne 30b
301: /* calculate the appended byte */
        clr r20
        mov r21, len0
        ldi r24, 0x80
        andi r21, 0x07
        breq 305f
        ld r20, X+
303:
        lsr r24
        dec r21
        brne 303b
305:
        or r20, r24
        st Z+, r20
        tst r23
        breq 32f
31: st Z+, r1
        dec r23
        brne 31b
32:
/*      if(length_b+1>64*8-64){ ; = 64*7-1 = 447 max(lengt
h_b)=511
                bmw_small_nextBlock(ctx, pctx.buffer);
                memset(pctx.buffer, 0, 64-8);
                ctx->counter -= 1;
        }
*/
        tst len1
        breq 400f
        cpi len0, 192
        brlo 400f
        movw r24, ctx0
        movw r22, buf0
        rcall bmw_small_nextBlock
        movw r26, buf0
        ldi r20, 64-8
350:
        st X+, r1
        dec r20
        brne 350b
        movw r30, ctx0
        adiw r30, 60
        ldd r21, Z+4
        ldd r22, Z+5
        ldd r23, Z+6
        ldd r24, Z+7
        subi r21, 1
        sbc r22, r1
        sbc r23, r1
        sbc r24, r1
        rjmp 410f
/*      *((uint64_t*)&(pctx.buffer[64-8])) = (uint64_t)(ct
x->counter*512LL)+(uint64_t)length_b;
        bmw_small_nextBlock(ctx, pctx.buffer);
*/
400:
        movw r30, ctx0
        adiw r30, 60
        ldd r21, Z+4
        ldd r22, Z+5
        ldd r23, Z+6
        ldd r24, Z+7
410:
        clr r25
        lsl r21
        rol r22
        rol r23
        rol r24
        rol r25
        mov r20, len0
        add r21, len1
        adc r22, r1
        adc r23, r1
        adc r24, r1
        adc r25, r1
        movw r30, buf0
        adiw r30, 64-8
        st Z+, r20
        st Z+, r21
        st Z+, r22
        st Z+, r23
        st Z+, r24
        st Z+, r25
        st Z+, r1
        st Z+, r1
        movw r24, ctx0
        movw r22, buf0
        rcall bmw_small_nextBlock
/*      memset(pctx.buffer, 0xaa, 64);
        for(i=0; i<16;++i){
                pctx.buffer[i*4] = i+0xa0;
        }
*/
        ldi r18, 0xa0
        ldi r19, 0xaa
        movw r26, buf0
500:
        st X+, r18
        st X+, r19
        st X+, r19
        st X+, r19
        inc r18
        sbrs r18, 4
        rjmp 500b
/*      bmw_small_nextBlock((bmw_small_ctx_t*)&pctx, ctx->
h);
        memcpy(ctx->h, pctx.buffer, 64);
*/
        movw r24, buf0
        movw r22, ctx0
        rcall bmw_small_nextBlock
        ldi r18, 64
        movw r26, ctx0
        movw r30, buf0
600:
        ld r20, Z+
        st X+, r20
        dec r18
        brne 600b

        stack_free_large 68
        pop_range 28, 29
        pop_range 2, 7
        ret


/*******************************************************
*********************
* void bmw224_ctx2hash(void* dest, const bmw224_ctx_t* ctx
){
*       memcpy(dest, &(ctx->h[9]), 224/8);
* }
*
* param dest:  r24:r25
* param ctx:   r22:r23
*/
.global bmw224_ctx2hash
bmw224_ctx2hash:
        movw r26, r24
        movw r30, r22
        adiw r30, 9*4
        ldi r22, 28
        rjmp 1f

/*******************************************************
*********************
* void bmw256_ctx2hash(void* dest, const bmw256_ctx_t* ctx
){
*       memcpy(dest, &(ctx->h[8]), 256/8);
* }
*
* param dest:  r24:r25
* param ctx:   r22:r23
*/
.global bmw256_ctx2hash
bmw256_ctx2hash:
        movw r26, r24
        movw r30, r22
        adiw r30, 8*4
        ldi r22, 32
1:
        ld r23, Z+
        st X+, r23
        dec r22
        brne 1b
        ret

/*******************************************************
*********************
* void bmw256(void* dest, const void* msg, uint32_t length
_b){
*       bmw_small_ctx_t ctx;
*       bmw256_init(&ctx);
*       while(length_b>=BMW_SMALL_BLOCKSIZE){
*               bmw_small_nextBlock(&ctx, msg);
*               length_b -= BMW_SMALL_BLOCKSIZE;
*               msg = (uint8_t*)msg + BMW_SMALL_BLOCKSIZE_
B;
*       }
*       bmw_small_lastBlock(&ctx, msg, length_b);
*       bmw256_ctx2hash(dest, &ctx);
* }
*
* param dest:     r24:r25
* param msg:      r22:r23
* param length_b: r18:r21
*/
ctx0 =   2
ctx1 =   3
```

```
msg0 =   4
msg1 =   5
len0 =   6
len1 =   7
len2 =   8
len3 =   9
dst0 =  10
dst1 =  11
.global bmw256
bmw256:
        push r16
        ldi r16, 1
        rjmp bmw_small_all

/*********************************************************
*********************
* void bmw224(void* dest, const void* msg, uint32_t length
_b){
*       bmw_small_ctx_t ctx;
*       bmw224_init(&ctx);
*       while(length_b>=BMW_SMALL_BLOCKSIZE){
*               bmw_small_nextBlock(&ctx, msg);
*               length_b -= BMW_SMALL_BLOCKSIZE;
*               msg = (uint8_t*)msg + BMW_SMALL_BLOCKSIZE_
B;
*       }
*       bmw_small_lastBlock(&ctx, msg, length_b);
*       bmw224_ctx2hash(dest, &ctx);
* }
*
* param dest:     r24:r25
* param msg:      r22:r23
* param length_b: r18:r21
*/
ctx0 =   2
ctx1 =   3
msg0 =   4
msg1 =   5
len0 =   6
len1 =   7
len2 =   8
len3 =   9
dst0 =  10
dst1 =  11
.global bmw224
bmw224:
        push r16
        clr r16

bmw_small_all:
        push_range 2, 11
        stack_alloc_large 64+4
        adiw r30, 1
        movw ctx0, r30
        movw dst0, r24
        movw msg0, r22
        movw len0, r18
        movw len2, r20
        movw r24, ctx0
        ldi r30, pm_lo8(init_lut)
        ldi r31, pm_hi8(init_lut)
        add r30, r16
        adc r31, r1
        icall
20:
        mov r18, len2
        or  r18, len3
        breq 50f
        movw r24, ctx0
        movw r22, msg0
        rcall bmw_small_nextBlock
        ldi r20, 2
        sub len1, r20
        sbc len2, r1
        sbc len3, r1
        ldi r20, 64
        add msg0, r20
        adc msg1, r1
        rjmp 20b
50:
        movw r24, ctx0
        movw r22, msg0
        movw r20, len0
        rcall bmw_small_lastBlock
        movw r24, dst0
        movw r22, ctx0
        ldi r30, pm_lo8(c2h_lut)
        ldi r31, pm_hi8(c2h_lut)
        add r30, r16
        adc r31, r1
        icall
        stack_free_large 64+4
        pop_range 2, 11
        pop r16
        ret

init_lut:
        rjmp bmw224_init
        rjmp bmw256_init
c2h_lut:
        rjmp bmw224_ctx2hash
        rjmp bmw256_ctx2hash

/*********************************************************
*********************
* void bmw224_init(bmw224_ctx_t* ctx){
*       uint8_t i;
*       ctx->h[0] = 0x00010203;
*       for(i=1; i<16; ++i){
*               ctx->h[i] = ctx->h[i-1]+ 0x04040404;
*       }
*       ctx->counter=0;
* }
*
* param ctx:  r24:r25
*/
.global bmw224_init
bmw224_init:
        movw r26, r24
        ldi r22, 0x03
        ldi r23, 0x02
        ldi r24, 0x01
        ldi r25, 0x00
bmw_small_init:
        st X+, r22
        st X+, r23
        st X+, r24
        st X+, r25
        ldi r18, 16-1
        ldi r20, 0x04
1:
        add r22, r20
        adc r23, r20
        adc r24, r20
        adc r25, r20
        st X+, r22
        st X+, r23
        st X+, r24
        st X+, r25
        dec r18
        brne 1b
        st X+, r1
        st X+, r1
        st X+, r1
        st X+, r1
        ret

.global bmw256_init
bmw256_init:
        movw r26, r24
        ldi r22, 0x43
        ldi r23, 0x42
        ldi r24, 0x41
        ldi r25, 0x40
        rjmp bmw_small_init


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

#if DEBUG

printQ:
        push_range 20, 25
        ldi r16, 4
        mov r9, r16
        movw r16, r24
        ldi r24, lo8(qdbg_str)
        ldi r25, hi8(qdbg_str)
        call cli_putstr_P
        clr r8
10:     ldi r24, lo8(qdbg_str1)
        ldi r25, hi8(qdbg_str1)
        call cli_putstr_P
        mov r24, r8
        call cli_hexdump_byte
        ldi r24, lo8(qdbg_str2)
        ldi r25, hi8(qdbg_str2)
        call cli_putstr_P
        movw r24, r16
        clr r23
        ldi r22, 4
        call cli_hexdump_rev
        add r16, r9
        adc r17, r1
        inc r8
        sbrs r8, 5
        rjmp 10b
        pop_range 20, 25
        ret
qdbg_str:  .asciz "\r\nDBG Q: "
qdbg_str1: .asciz "\r\n Q["
qdbg_str2: .asciz "] =  "


printX:
        push_range 6, 9
        push_range 16, 27
        push_range 30, 31
        ldi r16, 4
        mov r6, r22
```

```
	mov r9, r16
	movw r16, r24
	ldi r24, lo8(Xdbg_str)
	ldi r25, hi8(Xdbg_str)
	call cli_putstr_P
	mov r24, r6
	call cli_putc
	ldi r24, ':'
	call cli_putc
	clr r8
10:	ldi r24, lo8(Xdbg_str1)
	ldi r25, hi8(Xdbg_str1)
	call cli_putstr_P
	mov r24, r6
	call cli_putc
	ldi r24, '['
	call cli_putc
	mov r24, r8
	call cli_hexdump_byte
	ldi r24, lo8(Xdbg_str2)
	ldi r25, hi8(Xdbg_str2)
	call cli_putstr_P
	movw r24, r16
	clr r23
	ldi r22, 4
	call cli_hexdump_rev
	add r16, r9
	adc r17, r1
	inc r8
	sbrs r8, 4
	rjmp 10b
	pop_range 30, 31
	pop_range 16, 27
	pop_range 6, 9
	ret
Xdbg_str:  .asciz "\r\nDBG "
Xdbg_str1: .asciz "\r\n "
Xdbg_str2: .asciz "] = "

print32:
	push_range 6, 9
	push_range 16, 27
	push_range 30, 31
	movw r6, r22
	movw r8, r24
	ldi r24, lo8(Xdbg_str)
	ldi r25, hi8(Xdbg_str)
	call cli_putstr_P
	mov r24, r9
	call cli_hexdump_byte
	mov r24, r8
	call cli_hexdump_byte
	mov r24, r7
	call cli_hexdump_byte
	mov r24, r6
	call cli_hexdump_byte
	pop_range 30, 31
	pop_range 16, 27
	pop_range 6, 9
	ret


print_acc:
	push_range 16, 27
	push_range 30, 31
	ldi r24, lo8(Xdbg_str)
	ldi r25, hi8(Xdbg_str)
	call cli_putstr_P
	mov r24, r9
	call cli_hexdump_byte
	mov r24, r8
	call cli_hexdump_byte
	mov r24, r15
	call cli_hexdump_byte
	mov r24, r14
	call cli_hexdump_byte
	pop_range 30, 31
	pop_range 16, 27
	ret

#endif
```