아래는 csd_asm.S 코드입니다.

```
#include "uart_stack.s"


.extern csd_main

.align 8

// Our interrupt vector table
csd_entry:
        b csd_reset
        b .
        b .
        b .
        b .
        b .
        b csd_irq
        b .

.global main
csd_reset:
main:

//////////////////////////////////////////////////////////////////////
//                                                              //
//      [debugging macro]                                           //
//                                                              //
//    stmea r13, {r0-r15}                                       //
//    // store register value onto the stack (no sp changes)   //
//        uart_stack // execute debugging macro                       //
//        sub sp, sp, #64                                         //
//        ldr r0, [sp]                                          //
//        add sp, sp, #4                                          //
//        ldr r1, [sp]                                          //
//        add sp, sp, #4                                          //
//        ldr r2, [sp]                                          //
//        add sp, sp, #4                                          //
//        ldr r3, [sp]                                          //
//        add sp, sp, #4                                          //
//        ldr r4, [sp]                                          //
//        add sp, sp, #4                                          //
//        ldr r5, [sp]                                          //
//        add sp, sp, #4                                          //
//        ldr r6, [sp]                                          //
//        add sp, sp, #4                                          //
//        ldr r7, [sp]                                          //
//        add sp, sp, #4                                          //
//        ldr r8, [sp]                                          //
//        add sp, sp, #4                                          //
//        ldr r9, [sp]                                          //
//        add sp, sp, #4                                          //
//        ldr r10, [sp]                                         //
//        add sp, sp, #4                                          //
//        ldr r11, [sp]                                         //
//        add sp, sp, #4                                          //
//        ldr r12, [sp]                                         //
//       sub sp, sp, #48                                          //
//                                                              //
//////////////////////////////////////////////////////////////////////

        // sorting algorithm: selection sort (compare each value and select the
smallest)
```

```
    // 1st place
    stmea r13, {r0-r15}
uart_stack
sub sp, sp, #64
ldr r0, [sp]
add sp, sp, #4
ldr r1, [sp]
add sp, sp, #4
ldr r2, [sp]
add sp, sp, #4
ldr r3, [sp]
add sp, sp, #4
ldr r4, [sp]
add sp, sp, #4
ldr r5, [sp]
add sp, sp, #4
ldr r6, [sp]
add sp, sp, #4
ldr r7, [sp]
add sp, sp, #4
ldr r8, [sp]
add sp, sp, #4
ldr r9, [sp]
add sp, sp, #4
ldr r10, [sp]
add sp, sp, #4
ldr r11, [sp]
add sp, sp, #4
ldr r12, [sp]
add sp, sp, #8
ldr r14, [sp]
sub sp, sp, #56 // at this point, r0-r14 are returned

    ldr r0, =Input_data // store Input_data memory address in r0

    // 2nd place
    stmea r13, {r0-r15}
uart_stack
sub sp, sp, #64
ldr r0, [sp]
add sp, sp, #4
ldr r1, [sp]
add sp, sp, #4
ldr r2, [sp]
add sp, sp, #4
ldr r3, [sp]
add sp, sp, #4
ldr r4, [sp]
add sp, sp, #4
ldr r5, [sp]
add sp, sp, #4
ldr r6, [sp]
add sp, sp, #4
ldr r7, [sp]
add sp, sp, #4
ldr r8, [sp]
add sp, sp, #4
ldr r9, [sp]
add sp, sp, #4
ldr r10, [sp]
add sp, sp, #4
ldr r11, [sp]
add sp, sp, #4
ldr r12, [sp]
add sp, sp, #8
```

```
ldr r14, [sp]
sub sp, sp, #56

    ldr r1, =Output_data // store Output_data memory address in r1

    // 3rd place
    stmea r13, {r0-r15}
uart_stack
sub sp, sp, #64
ldr r0, [sp]
add sp, sp, #4
ldr r1, [sp]
add sp, sp, #4
ldr r2, [sp]
add sp, sp, #4
ldr r3, [sp]
add sp, sp, #4
ldr r4, [sp]
add sp, sp, #4
ldr r5, [sp]
add sp, sp, #4
ldr r6, [sp]
add sp, sp, #4
ldr r7, [sp]
add sp, sp, #4
ldr r8, [sp]
add sp, sp, #4
ldr r9, [sp]
add sp, sp, #4
ldr r10, [sp]
add sp, sp, #4
ldr r11, [sp]
add sp, sp, #4
ldr r12, [sp]
add sp, sp, #8
ldr r14, [sp]
sub sp, sp, #56

    mov r2, #-4 // r2 stands for start position

    // 4th place
    stmea r13, {r0-r15}
uart_stack
sub sp, sp, #64
ldr r0, [sp]
add sp, sp, #4
ldr r1, [sp]
add sp, sp, #4
ldr r2, [sp]
add sp, sp, #4
ldr r3, [sp]
add sp, sp, #4
ldr r4, [sp]
add sp, sp, #4
ldr r5, [sp]
add sp, sp, #4
ldr r6, [sp]
add sp, sp, #4
ldr r7, [sp]
add sp, sp, #4
ldr r8, [sp]
add sp, sp, #4
ldr r9, [sp]
add sp, sp, #4
ldr r10, [sp]
```

```
    add sp, sp, #4
    ldr r11, [sp]
    add sp, sp, #4
    ldr r12, [sp]
    add sp, sp, #8
    ldr r14, [sp]
    sub sp, sp, #56

first_loop:

        add r2, r2, #4 // assign r2 the first start position

        // 5th place
        stmea r13, {r0-r15}
    uart_stack
    sub sp, sp, #64
    ldr r0, [sp]
    add sp, sp, #4
    ldr r1, [sp]
    add sp, sp, #4
    ldr r2, [sp]
    add sp, sp, #4
    ldr r3, [sp]
    add sp, sp, #4
    ldr r4, [sp]
    add sp, sp, #4
    ldr r5, [sp]
    add sp, sp, #4
    ldr r6, [sp]
    add sp, sp, #4
    ldr r7, [sp]
    add sp, sp, #4
    ldr r8, [sp]
    add sp, sp, #4
    ldr r9, [sp]
    add sp, sp, #4
    ldr r10, [sp]
    add sp, sp, #4
    ldr r11, [sp]
    add sp, sp, #4
    ldr r12, [sp]
    add sp, sp, #8
    ldr r14, [sp]
    sub sp, sp, #56

        mov r3, r2 // r3 stands for the position of the smallest value

        // 6th place
        stmea r13, {r0-r15}
    uart_stack
    sub sp, sp, #64
    ldr r0, [sp]
    add sp, sp, #4
    ldr r1, [sp]
    add sp, sp, #4
    ldr r2, [sp]
    add sp, sp, #4
    ldr r3, [sp]
    add sp, sp, #4
    ldr r4, [sp]
    add sp, sp, #4
    ldr r5, [sp]
    add sp, sp, #4
    ldr r6, [sp]
    add sp, sp, #4
    ldr r7, [sp]
    add sp, sp, #4
    ldr r8, [sp]
    add sp, sp, #4
    ldr r9, [sp]
    add sp, sp, #4
    ldr r10, [sp]
    add sp, sp, #4
    ldr r11, [sp]
    add sp, sp, #4
    ldr r12, [sp]
    add sp, sp, #8
    ldr r14, [sp]
    sub sp, sp, #56

        mov r4, r3 // r4 stands for compared position

        // 7th place
        stmea r13, {r0-r15}
    uart_stack
    sub sp, sp, #64
    ldr r0, [sp]
    add sp, sp, #4
    ldr r1, [sp]
    add sp, sp, #4
    ldr r2, [sp]
    add sp, sp, #4
    ldr r3, [sp]
    add sp, sp, #4
    ldr r4, [sp]
    add sp, sp, #4
    ldr r5, [sp]
    add sp, sp, #4
    ldr r6, [sp]
    add sp, sp, #4
    ldr r7, [sp]
    add sp, sp, #4
    ldr r8, [sp]
    add sp, sp, #4
    ldr r9, [sp]
    add sp, sp, #4
    ldr r10, [sp]
    add sp, sp, #4
    ldr r11, [sp]
    add sp, sp, #4
    ldr r12, [sp]
    add sp, sp, #8
    ldr r14, [sp]
    sub sp, sp, #56

second_loop:

        add r4, r4, #4 // r4 stands for the next position of r2

        // 8th place
        stmea r13, {r0-r15}
    uart_stack
    sub sp, sp, #64
    ldr r0, [sp]
    add sp, sp, #4
    ldr r1, [sp]
    add sp, sp, #4
    ldr r2, [sp]
    add sp, sp, #4
    ldr r3, [sp]
```

```
add sp, sp, #4
ldr r4, [sp]
add sp, sp, #4
ldr r5, [sp]
add sp, sp, #4
ldr r6, [sp]
add sp, sp, #4
ldr r7, [sp]
add sp, sp, #4
ldr r8, [sp]
add sp, sp, #4
ldr r9, [sp]
add sp, sp, #4
ldr r10, [sp]
add sp, sp, #4
ldr r11, [sp]
add sp, sp, #4
ldr r12, [sp]
add sp, sp, #8
ldr r14, [sp]
sub sp, sp, #56

    ldr r5, [r0, r3] // load the word which needs to be compared into r5

    // 9th place
    stmea r13, {r0-r15}
uart_stack
sub sp, sp, #64
ldr r0, [sp]
add sp, sp, #4
ldr r1, [sp]
add sp, sp, #4
ldr r2, [sp]
add sp, sp, #4
ldr r3, [sp]
add sp, sp, #4
ldr r4, [sp]
add sp, sp, #4
ldr r5, [sp]
add sp, sp, #4
ldr r6, [sp]
add sp, sp, #4
ldr r7, [sp]
add sp, sp, #4
ldr r8, [sp]
add sp, sp, #4
ldr r9, [sp]
add sp, sp, #4
ldr r10, [sp]
add sp, sp, #4
ldr r11, [sp]
add sp, sp, #4
ldr r12, [sp]
add sp, sp, #8
ldr r14, [sp]
sub sp, sp, #56

    ldr r6, [r0, r4] // load the next word which needs to be compared into r6

    // 10th place
    stmea r13, {r0-r15}
uart_stack
sub sp, sp, #64
ldr r0, [sp]
add sp, sp, #4
```

```
    ldr r1, [sp]
    add sp, sp, #4
    ldr r2, [sp]
    add sp, sp, #4
    ldr r3, [sp]
    add sp, sp, #4
    ldr r4, [sp]
    add sp, sp, #4
    ldr r5, [sp]
    add sp, sp, #4
    ldr r6, [sp]
    add sp, sp, #4
    ldr r7, [sp]
    add sp, sp, #4
    ldr r8, [sp]
    add sp, sp, #4
    ldr r9, [sp]
    add sp, sp, #4
    ldr r10, [sp]
    add sp, sp, #4
    ldr r11, [sp]
    add sp, sp, #4
    ldr r12, [sp]
    add sp, sp, #8
    ldr r14, [sp]
    sub sp, sp, #56

    cmp r5, r6 // compare

        movgt r3, r4
        // if r5 is greater than r6 (which means r6 is smaller) move value of r4 to
r3
        // it means the smallest value has been changed

        cmp r4, #124 // check the remaining number of comparison in
second_loop
        bne second_loop // if times are remaining, go to second_loop label and
continue comparison

    // if second_loop ends then store the smallest value into Output_data

        ldr r7, [r0, r3] // load the smallest word into r7
        str r7, [r1, r2] // store the word to Output_data

        // also move the value (which is not the smallest) to Input_data at empty
position

        ldr r7, [r0, r2] // load the not-smallest value into r7
        str r7, [r0, r3] // fill the empty place in Input_data (which was filled with
the smallest)

        cmp r2, #120 // check the remaining number of comparison in first_loop

        bne first_loop // if times are remaining, go to first_loop lable and continue
switching

        // after escaping from the first_loop, only one value (the biggest) left in
Input_data

        ldr r7, [r0, #124] // load the biggest value into r7
        str r7, [r1, #124] // fill the list completely (store the biggest value to the
list)

        b csd_main // go to csd_main which leads to program ends
```

```
.data
.align 4

Input_data: .word 2, 0, -7, -1, 3, 8, -4, 10
                        .word -9, -16, 15, 13, 1, 4, -3, 14
                        .word -8, -10, -15, 6, -13, -5, 9, 12
                        .word -11, -14, -6, 11, 5, 7, -2, -12

Output_data: .word 0, 0, 0, 0, 0, 0, 0, 0
                         .word 0, 0, 0, 0, 0, 0, 0, 0
                         .word 0, 0, 0, 0, 0, 0, 0, 0
                         .word 0, 0, 0, 0, 0, 0, 0, 0

// Normal Interrupt Service Routine
csd_irq:
        b .
```

아래는 uart_regs.h 코드입니다.

```
/////////////////////////////////////////////////////
#define slcr_UART_RST_CTRL                         0xF8000228
#define slcr_UART_CLK_CTRL                         0xF8000154

#define UART1_BASE                                       0xE0001000
#define UART_CONTROL_REG0_OFFSET              0x0
#define UART_MODE_REG0_OFFSET                      0x4
#define UART_INTRPT_EN_REG0_OFFSET                 0x8
#define UART_INTRPT_DIS_REG0_OFFSET                0xC
#define UART_INTRPT_MASK_REG0_OFFSET          0x10
#define UART_BAUD_RATE_GEN_REG0_OFFSET        0x18
#define UART_RCVR_TIMEOUT_REG0_OFFSET         0x1C
#define UART_RCVR_FIFO_TRG_LV0_OFFSET         0x20
#define UART_CHANNEL_STS_REG0_OFFSET          0x2C
#define UART_TX_RX_FIFO0_OFFSET                    0x30
#define UART_BAUD_RATE_DIV_REG0_OFFSET        0x34

//////////////////////////////////////////////////////////////

#define uart_base                                        0xE0001000
#define uart_Control_reg0                         uart_base
#define uart_mode_reg0                            uart_base + 0x00000004
#define uart_Baud_rate_gen_reg0                   uart_base + 0x00000018
#define uart_Baud_rate_divider_reg0          uart_base + 0x00000034
#define uart_Modem_ctrl_reg0                 uart_base + 0x00000024
#define uart_Modem_sts_reg0                  uart_base + 0x00000028
#define uart_TX_RX_FIFO0                          uart_base + 0x00000030
#define uart_Channel_sts_reg0                uart_base + 0x0000002C
```

아래는 uart_init.s 코드입니다.

```
#include "uart_regs.h"

.macro uart_init

/*
# 1.Reset controller
        ldr             r0, =slcr_UART_RST_CTRL
        ldr             r1, [r0, #0]                @ read
slcr.UART_RST_CTRL
        orr             r1, r1, #0x0000000A         @ set both
slcr.UART_RST_CTRL[UART1_REF_RST, UART1_CPU1X_RST] (bit[3][1])
        str             r1, [r0, #0]                @ update

# 2.Configure I/O signal routing
# 3.Configure UART_Ref_Clk
        ldr             r0, =slcr_UART_CLK_CTRL
        ldr             r1, =0x00001402             @ write 0x00001402 to
slcr.UART_CLK_CTRL @ mov    r1, #0x00001402(*** ERROR ***)
        str             r1, [r0, #0]                @ update
*/

# 4.Configure controller functions
        ldr             r0, =UART1_BASE

#       4-1. Configure UART character frame
        mov             r1, #0x00000020
        str             r1, [r0, #UART_MODE_REG0_OFFSET]

#       4-2. Configure the Baud Rate
        # a-b. Disable Rx Path and Tx Path
        ldr             r1, [r0, #UART_CONTROL_REG0_OFFSET]      @ read
uart.Control_reg0
        bic             r1, r1, #0x0000003C                          @
clear TXDIS, TXEN, RXDIS, RXEN (bit[5][4][3][2])
        orr             r1, r1, #0x00000028                          @
TXDIS = 1 TXEN = 0 RXDIS = 1 RXEN = 0 (bit[5][4][3][2])
        str             r1, [r0, #UART_CONTROL_REG0_OFFSET]      @ update
        # c-d. Write the calculated CD value and BDIV
        mov             r1, #0x0000003E
                        @ CD = 62 (Baud rate 115200)
        str             r1, [r0, #UART_BAUD_RATE_GEN_REG0_OFFSET]    @
update uart.Baud_rate_gen_reg0
        mov             r1, #0x00000006
                @ BDIV = 6 (Baud rate 115200)
        str             r1, [r0, #UART_BAUD_RATE_DIV_REG0_OFFSET]    @
update uart.Baud_rate_divider_reg0 @ strb     r1, [r0,
#UART_BAUD_RATE_DIV_REG0_OFFSET]
        # e. Reset Tx and Px Path
        ldr             r1, [r0, #UART_CONTROL_REG0_OFFSET]      @ read
uart.Control_reg0
        orr             r1, r1, #0x00000003                          @
set TXRST, RXRST (bit[1][0]:self-clearing) - this resets Tx and Rx paths
        str             r1, [r0, #UART_CONTROL_REG0_OFFSET]      @ update
        # f-g. Enable Rx Path and Tx Path
        ldr             r1, [r0, #UART_CONTROL_REG0_OFFSET]      @ read
uart.Control_reg0
        bic             r1, r1, #0x0000003C                          @
clear TXDIS, TXEN, RXDIS, RXEN (bit[5][4][3][2])
        orr             r1, r1, #0x00000014                          @
TXDIS = 0 TXEN = 1 RXDIS = 0 RXEN = 1 (bit[5][4][3][2])
        str             r1, [r0, #UART_CONTROL_REG0_OFFSET]      @ update
```

```
#	4-5. Enable the Controller
	ldr		r1, =0x00000117					@
write 0x00000117 to uart.Control_reg0
	str		r1, [r0, #UART_CONTROL_REG0_OFFSET]	@ update

.endm
```

아래는 uart_stack.s 코드입니다.

```
.data
string:
        .ascii
"------------------------------------------------------------------------------------
"
        .byte 0x0D
        .byte 0x0A
        .ascii "r0 = 0x"
        .byte 0x00

string_1:
        .ascii "_"
        .byte 0x00

string_2:
        .ascii ", r1 = 0x"
        .byte 0x00

string_3:
        .ascii ", r2 = 0x"
        .byte 0x00

string_4:
        .ascii ", r3 = 0x"
        .byte 0x00

string_5:
        .byte 0x0D
        .byte 0x0A
        .ascii "r4 = 0x"
        .byte 0x00

string_6:
        .ascii ", r5 = 0x"
        .byte 0x00

string_7:
        .ascii ", r6 = 0x"
        .byte 0x00

string_8:
        .ascii ", r7 = 0x"
        .byte 0x00

string_9:
        .byte 0x0D
        .byte 0x0A
        .ascii "r8 = 0x"
        .byte 0x00

string_10:
        .ascii ", r9 = 0x"
        .byte 0x00

string_11:
        .ascii ", r10 = 0x"
        .byte 0x00

string_12:
        .ascii ", r11 = 0x"
        .byte 0x00
```

```
string_13:
        .byte 0x0D
        .byte 0x0A
        .ascii "r12 = 0x"
        .byte 0x00

string_14:
        .ascii ", r13 = 0x"
        .byte 0x00

string_15:
        .ascii ", r14 = 0x"
        .byte 0x00

string_16:
        .ascii ", r15 = 0x"
        .byte 0x00

string_nzcv:
        .byte 0x0D
        .byte 0x0A
        .ascii "cpsr = "
        .byte 0x00

string_if:
        .ascii ", "
        .byte 0x00

string_mode:
        .ascii " mode, current mode = "
        .byte 0x00

string_current_mode:
        .ascii " (=0x"
        .byte 0x00

string_final:
        .ascii ")"
        .byte 0x0D
        .byte 0x0A
        .ascii
"-----------------------------------------------------------------------
"
        .byte 0x0D
        .byte 0x0A
        .byte 0x0A
        .byte 0x00

.text

.macro uart_stack

        uart_init
        bl uart_print

.endm


#include "uart_init.s"
#include "uart_regs.h"

//-----------------------------------------------------------------------
-
// r0 = 0x1000_0123, r1 = 0xffee_0112, r2 = 0x9800_ab00, r3 = 0xfe03_0010
// r4 = 0xffff_1000, r5 = 0xc123_0112, r6 = 0x1800_1100, r7 = 0xbe10_0030
// r8 = 0x2200_0140, r9 = 0x55ee_0112, r10 = 0x1200_1200, r11 = 0x9803_2210
// r12 = 0x3300_0100, r13 = 0xccee_0112, r14 = 0x3400_ab00, r15 = 0x0010_0304
// cpsr = nZCv, IF, ARM mode, current mode = SVC ( =0x6000_00d3)
//---------------------------------------------------------------------------
-

uart_print:

        add sp, sp, #60
        ldr r5, [sp] // stack contains pc register value already added by 8
        sub r5, r5, #8 // it should be decremented by 8
        stmea sp, {r5}
        sub sp, sp, #60

        mrs r4, cpsr // since stack doesn't contain cpsr register value
        add sp, sp, #64
        stmea sp, {r4} // store the cpsr register value onto the stack
        sub sp, sp, #64

        mov r8, #0

        ldr r1, =string

        ldr r1, =string
        .ltorg

TRANSMIT_loop:

        // ---------  Check to see if the Tx FIFO is empty
------------------------------
        ldr     r2, [r0, #0x2C]  @ get Channel Status Register
        and     r2, r2, #0x8            @ get Transmit Buffer Empty bit(bit[3:3])
        cmp     r2, #0x8                                @ check if TxFIFO is empty
and ready to receive new data
        bne     TRANSMIT_loop           @ if TxFIFO is NOT empty, keep checking
until it is empty

//---------------------------------------------------------------------------
---

        ldrb     r3, [r1], #1
        strb            r3, [r0, #0x30]  @ fill the TxFIFO with 0x48
        cmp      r3, #0x00
        bne             TRANSMIT_loop

        mov r5, #0
        add sp, sp, #4 // stack에 0x12345678의 경우 78, 56, 34, 12 이런 식으로
저장되어 있음

TRANSMIT_loop_1:

        // ---------  Check to see if the Tx FIFO is empty
------------------------------
        ldr     r2, [r0, #0x2C]  @ get Channel Status Register
        and     r2, r2, #0x8            @ get Transmit Buffer Empty bit(bit[3:3])
        cmp     r2, #0x8                                @ check if TxFIFO is empty
and ready to receive new data
        bne     TRANSMIT_loop_1                 @ if TxFIFO is NOT empty, keep
checking until it is empty

//---------------------------------------------------------------------------
---

        sub sp, sp, #1
        ldrb     r3, [sp] // register의 상위 2 bit가 출력되기 위함
        mov r6, r3, LSR #4 // 상위 2 bit 중 상위의 bit가 r6에 저장됨
        and r7, r3, #15 // 상위 2 bit 중 하위의 bit가 r7에 저장됨

        cmp r6, #9
        addle r6, r6, #48 // if r6 is less than or equal to 9, then add 48 (based on
ASCII)
        addgt r6, r6, #87 // if r6 is greater than 9, then add 87 (based on ASCII)
        strb            r6, [r0, #0x30]  @ fill the TxFIFO with 0x48

        cmp r7, #9
        addle r7, r7, #48 // if r7 is less than or equal to 9, then add 48 (based on
ASCII)
        addgt r7, r7, #87 // if r7 is greater than 9, then add 87 (based on ASCII)
        strb            r7, [r0, #0x30]  @ fill the TxFIFO with 0x48

        add r5, r5, #1
        cmp r5, #2
        bne             TRANSMIT_loop_1

        mov r3, #95 // r3 has 8-bit value whereas 4-bit is needed
        strb            r3, [r0, #0x30]  @ fill the TxFIFO with 0x48

TRANSMIT_loop_2:

        // ---------  Check to see if the Tx FIFO is empty
------------------------------
        ldr     r2, [r0, #0x2C]  @ get Channel Status Register
        and     r2, r2, #0x8            @ get Transmit Buffer Empty bit(bit[3:3])
        cmp     r2, #0x8                                @ check if TxFIFO is empty
and ready to receive new data
        bne     TRANSMIT_loop_2                 @ if TxFIFO is NOT empty, keep
checking until it is empty

//----------------------------------------------------------------------
---

        sub sp, sp, #1
        ldrb      r3, [sp]
        mov r6, r3, LSR #4
        and r7, r3, #15

        cmp r6, #9
        addle r6, r6, #48 // if r6 is less than or equal to 9, then add 48 (based on
ASCII)
        addgt r6, r6, #87 // if r6 is greater than 9, then add 87 (based on ASCII)
        strb            r6, [r0, #0x30]  @ fill the TxFIFO with 0x48

        cmp r7, #9
        addle r7, r7, #48 // if r7 is less than or equal to 9, then add 48 (based on
ASCII)
        addgt r7, r7, #87 // if r7 is greater than 9, then add 87 (based on ASCII)
        strb            r7, [r0, #0x30]  @ fill the TxFIFO with 0x48

        add r5, r5, #1
        cmp r5, #4
        bne             TRANSMIT_loop_2

        add r8, r8, #1

        cmp r8, #1
        beq print_r1
        cmp r8, #2
        beq print_r2
        cmp r8, #3
        beq print_r3
        cmp r8, #4
        beq print_r4
        cmp r8, #5
        beq print_r5
        cmp r8, #6
        beq print_r6
        cmp r8, #7
        beq print_r7
        cmp r8, #8
        beq print_r8
        cmp r8, #9
        beq print_r9
        cmp r8, #10
        beq print_r10
        cmp r8, #11
        beq print_r11
        cmp r8, #12
        beq print_r12
        cmp r8, #13
        beq print_r13
        cmp r8, #14
        beq print_r14
        cmp r8, #15
        beq print_r15
        cmp r8, #16
        beq print_nzcv // goto print_nzcv to print cpsr

print_r1:
        ldr r1, =string_2
        add sp, sp, #4
        b TRANSMIT_loop

print_r2:
        ldr r1, =string_3
        add sp, sp, #4
        b TRANSMIT_loop

print_r3:
        ldr r1, =string_4
        add sp, sp, #4
        b TRANSMIT_loop

print_r4:
        ldr r1, =string_5
        add sp, sp, #4
        b TRANSMIT_loop

print_r5:
        ldr r1, =string_6
        add sp, sp, #4
        b TRANSMIT_loop

print_r6:
        ldr r1, =string_7
        add sp, sp, #4
        b TRANSMIT_loop

print_r7:
        ldr r1, =string_8
        add sp, sp, #4
        b TRANSMIT_loop

print_r8:
        ldr r1, =string_9
        add sp, sp, #4
        b TRANSMIT_loop

print_r9:
        ldr r1, =string_10
        add sp, sp, #4
        b TRANSMIT_loop

print_r10:
        ldr r1, =string_11
        add sp, sp, #4
        b TRANSMIT_loop

print_r11:
        ldr r1, =string_12
        add sp, sp, #4
        b TRANSMIT_loop

print_r12:
        ldr r1, =string_13
        add sp, sp, #4
        b TRANSMIT_loop

print_r13:
        ldr r1, =string_14
        add sp, sp, #4
        b TRANSMIT_loop

print_r14:
        ldr r1, =string_15
        add sp, sp, #4
        b TRANSMIT_loop

print_r15:
        ldr r1, =string_16
        add sp, sp, #4
        b TRANSMIT_loop

// cpsr = nZCv, IF, ARM mode, current mode = SVC ( =0x6000_00d3)

print_nzcv:

        ldr r1, =string_nzcv

        loop:

        // ---------  Check to see if the Tx FIFO is empty
-------------------------------
        ldr     r2, [r0, #0x2C] @ get Channel Status Register
        and     r2, r2, #0x8            @ get Transmit Buffer Empty bit(bit[3:3])
        cmp     r2, #0x8                                @ check if TxFIFO is empty
and ready to receive new data
        bne     loop            @ if TxFIFO is NOT empty, keep checking until it is
empty

//--------------------------------------------------------------------------
---

        ldrb      r3, [r1], #1
        strb              r3, [r0, #0x30]  @ fill the TxFIFO with 0x48
        cmp       r3, #0x00
        bne               loop

        add sp, sp, #8 // sp points to the place where cpsr is stored

        loop_1:
```

```
        // ---------  Check to see if the Tx FIFO is empty
------------------------------
        ldr     r2, [r0, #0x2C] @ get Channel Status Register
        and     r2, r2, #0x8            @ get Transmit Buffer Empty bit(bit[3:3])
        cmp     r2, #0x8                                @ check if TxFIFO is empty
and ready to receive new data
        bne     loop_1          @ if TxFIFO is NOT empty, keep checking until it is
empty

//------------------------------------------------------------------------------
---

        sub sp, sp, #1
        ldrb     r3, [sp] // register의 상위 2 bit가 출력되기 위함
        mov r6, r3, LSR #4 // 상위 2 bit 중 상위의 bit가 r6에 저장됨
        mov r7, #0

        // cpsr = nzcv
        cmp r6, #0
        moveq r7, #110 // 110 imples small alphabet n
        streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r7, #122 // 122 imples small alphabet z
        streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r7, #99 // 99 imples small alphabet c
        streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r7, #118 // 118 imples small alphabet v
        streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48

        // cpsr = nzcV
        cmp r6, #1
        moveq r7, #110 // 110 imples small alphabet n
        streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r7, #122 // 122 imples small alphabet z
        streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r7, #99 // 99 imples small alphabet c
        streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r7, #86 // 86 imples large alphabet V
        streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48

        // cpsr = nzCv
        cmp r6, #2
        moveq r7, #110 // 110 imples small alphabet n
        streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r7, #122 // 122 imples small alphabet z
        streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r7, #67 // 67 imples large alphabet C
        streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r7, #118 // 118 imples small alphabet v
        streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48

        // cpsr = nzCV
        cmp r6, #3
        moveq r7, #110 // 110 imples small alphabet n
        streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r7, #122 // 122 imples small alphabet z
        streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r7, #67 // 67 imples large alphabet C
        streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r7, #86 // 86 imples large alphabet V
        streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48

        // cpsr = nZcv
        cmp r6, #4
        moveq r7, #110 // 110 imples small alphabet n
```

```
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #90 // 90 imples large alphabet Z
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #99 // 99 imples small alphabet c
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #118 // 118 imples small alphabet v
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48

// cpsr = nZcV
cmp r6, #5
moveq r7, #110 // 110 imples small alphabet n
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #90 // 90 imples large alphabet Z
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #99 // 99 imples small alphabet c
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #86 // 86 imples large alphabet V
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48

// cpsr = nZCv
cmp r6, #6
moveq r7, #110 // 110 imples small alphabet n
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #90 // 90 imples large alphabet Z
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #67 // 67 imples large alphabet C
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #118 // 118 imples small alphabet v
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48

// cpsr = nZCV
cmp r6, #7
moveq r7, #110 // 110 imples small alphabet n
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #90 // 90 imples large alphabet Z
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #67 // 67 imples large alphabet C
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #86 // 86 imples large alphabet V
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48

// cpsr = Nzcv
cmp r6, #8
moveq r7, #78 // 78 imples large alphabet N
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #122 // 122 imples small alphabet z
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #99 // 99 imples small alphabet c
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #118 // 118 imples small alphabet v
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48

// cpsr = NzcV
cmp r6, #9
moveq r7, #78 // 78 imples large alphabet N
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #122 // 122 imples small alphabet z
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #99 // 99 imples small alphabet c
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #86 // 86 imples large alphabet V
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48

// cpsr = NzCv
cmp r6, #10
```

```
moveq r7, #78 // 78 imples large alphabet N
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #122 // 122 imples small alphabet z
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #67 // 67 imples large alphabet C
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #118 // 118 imples small alphabet v
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48

// cpsr = NzCV
cmp r6, #11
moveq r7, #78 // 78 imples large alphabet N
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #122 // 122 imples small alphabet z
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #67 // 67 imples large alphabet C
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #86 // 86 imples large alphabet V
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48

// cpsr = NZcv
cmp r6, #12
moveq r7, #78 // 78 imples large alphabet N
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #90 // 90 imples large alphabet Z
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #99 // 99 imples small alphabet c
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #118 // 118 imples small alphabet v
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48

// cpsr = NZcV
cmp r6, #13
moveq r7, #78 // 78 imples large alphabet N
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #90 // 90 imples large alphabet Z
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #99 // 99 imples small alphabet c
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #86 // 86 imples large alphabet V
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48

// cpsr = NZCv
cmp r6, #14
moveq r7, #78 // 78 imples large alphabet N
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #90 // 90 imples large alphabet Z
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #67 // 67 imples large alphabet C
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #118 // 118 imples small alphabet v
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48

// cpsr = NZCV
cmp r6, #15
moveq r7, #78 // 78 imples large alphabet N
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #90 // 90 imples large alphabet Z
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #67 // 67 imples large alphabet C
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
moveq r7, #86 // 86 imples large alphabet V
streqb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48

ldr r1, =string_if
```

```
        loop_2:

        // ---------  Check to see if the Tx FIFO is empty
------------------------------
        ldr     r2, [r0, #0x2C] @ get Channel Status Register
        and     r2, r2, #0x8            @ get Transmit Buffer Empty bit(bit[3:3])
        cmp     r2, #0x8                                @ check if TxFIFO is empty
and ready to receive new data
        bne     loop_2          @ if TxFIFO is NOT empty, keep checking until it is
empty

//-----------------------------------------------------------------------------
---

        ldrb     r3, [r1], #1
        strb            r3, [r0, #0x30]  @ fill the TxFIFO with 0x48
        cmp      r3, #0x00
        bne             loop_2

        loop_3:

        // ---------  Check to see if the Tx FIFO is empty
------------------------------
        ldr     r2, [r0, #0x2C] @ get Channel Status Register
        and     r2, r2, #0x8            @ get Transmit Buffer Empty bit(bit[3:3])
        cmp     r2, #0x8                                @ check if TxFIFO is empty
and ready to receive new data
        bne     loop_3          @ if TxFIFO is NOT empty, keep checking until it is
empty

//-----------------------------------------------------------------------------
---

        sub sp, sp, #3 // sp points to the place where I bit and F bits are stored
        ldrb     r3, [sp]
        mov r6, r3, LSR #4 // r6 imples cpsr[7:4] bits

        // I bit = 0, F bit = 0
        cmp r6, #0
        movge r7, #105 // 105 imples small alphabet i
        movge r8, #102 // 102 imples small alphabet f

        // I bit = 0, F bit = 1
        cmp r6, #4
        movge r7, #105 // 105 imples small alphabet i
        movge r8, #70 // 70 imples large alphabet F

        // I bit = 1, F bit = 0
        cmp r6, #8
        movge r7, #73 // 73 imples large alphabet I
        movge r8, #102 // 102 imples small alphabet f

        // I bit = 1, F bit = 1
        cmp r6, #12
        movge r7, #73 // 73 imples large alphabet I
        movge r8, #70 // 70 imples large alphabet F

        strgeb          r7, [r0, #0x30]  @ fill the TxFIFO with 0x48
        strgeb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48

        ldr r1, =string_if

        loop_4:
```

```
        // ---------  Check to see if the Tx FIFO is empty
------------------------------
        ldr     r2, [r0, #0x2C] @ get Channel Status Register
        and     r2, r2, #0x8            @ get Transmit Buffer Empty bit(bit[3:3])
        cmp     r2, #0x8                                @ check if TxFIFO is empty
and ready to receive new data
        bne     loop_4          @ if TxFIFO is NOT empty, keep checking until it is
empty

//--------------------------------------------------------------------------
---

        ldrb     r3, [r1], #1
        strb            r3, [r0, #0x30]  @ fill the TxFIFO with 0x48
        cmp       r3, #0x00
        bne             loop_4

        loop_5:

        // ---------  Check to see if the Tx FIFO is empty
------------------------------
        ldr     r2, [r0, #0x2C] @ get Channel Status Register
        and     r2, r2, #0x8            @ get Transmit Buffer Empty bit(bit[3:3])
        cmp     r2, #0x8                                @ check if TxFIFO is empty
and ready to receive new data
        bne     loop_5          @ if TxFIFO is NOT empty, keep checking until it is
empty

//--------------------------------------------------------------------------
---

        add sp, sp, #3 // sp points to the place where J bit is stored
        ldrb     r3, [sp]
        and r7, r3, #1 // r7 implies cpsr[24] which is J bit

        sub sp, sp, #3 // sp points to the place where T bit is stored
        ldrb     r3, [sp]
        and r8, r3, #32 // r8 implies cpsr[5] which is T bit

        add r8, r8, r8
        add r7, r7, r8 // 1 J bit + 2 T bit can differentiate 4 modes

        // ARM mode
        cmp r7, #0
        moveq r8, #65 // 65 imples large alphabet A
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r8, #82 // 82 imples large alphabet R
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r8, #77 // 77 imples large alphabet M
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48

        // Thumb mode
        cmp r7, #2
        moveq r8, #84 // 84 imples large alphabet T
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r8, #104 // 104 imples small alphabet h
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r8, #117 // 117 imples small alphabet u
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r8, #109 // 109 imples small alphabet m
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r8, #98 // 98 imples small alphabet b
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48

        // Jazelle mode
        cmp r7, #1
        moveq r8, #74 // 74 imples large alphabet J
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r8, #97 // 97 imples small alphabet a
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r8, #122 // 122 imples small alphabet z
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r8, #101 // 101 imples small alphabet e
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r8, #108 // 108 imples small alphabet l
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r8, #108 // 108 imples small alphabet l
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r8, #101 // 101 imples small alphabet e
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48

        // ThumbEE mode
        cmp r7, #3
        moveq r8, #84 // 84 imples large alphabet T
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r8, #104 // 104 imples small alphabet h
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r8, #117 // 117 imples small alphabet u
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r8, #109 // 109 imples small alphabet m
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r8, #98 // 98 imples small alphabet b
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r8, #69 // 69 imples large alphabet E
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48
        moveq r8, #69 // 69 imples large alphabet E
        streqb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48

        ldr r1, =string_mode

        loop_6:

        // ---------  Check to see if the Tx FIFO is empty
------------------------------
        ldr      r2, [r0, #0x2C]  @ get Channel Status Register
        and      r2, r2, #0x8              @ get Transmit Buffer Empty bit(bit[3:3])
        cmp      r2, #0x8                               @ check if TxFIFO is empty
and ready to receive new data
        bne      loop_6          @ if TxFIFO is NOT empty, keep checking until it is
empty

//--------------------------------------------------------------------------
---

        ldrb      r3, [r1], #1
        strb            r3, [r0, #0x30]  @ fill the TxFIFO with 0x48
        cmp        r3, #0x00
        bne             loop_6

        ldrb      r3, [sp] // r3 imples cpsr[7:0]
        and r7, r3, #31 // r7 imples cpsr[5:0] bits (mode bits)

        // USR
        cmp r7, #16
        movge r8, #85 // 85 imples large alphabet U
        movge r9, #83 // 83 imples large alphabet S
        movge r10, #82 // 82 imples large alphabet R

        // FIQ
        cmp r7, #17
        movge r8, #70 // 70 imples large alphabet F
        movge r9, #73 // 73 imples large alphabet I
        movge r10, #81 // 81 imples large alphabet Q

        // IRQ
        cmp r7, #18
        movge r8, #73 // 73 imples large alphabet I
        movge r9, #82 // 82 imples large alphabet R
        movge r10, #81 // 81 imples large alphabet Q

        // SVC
        cmp r7, #19
        movge r8, #83 // 83 imples large alphabet S
        movge r9, #86 // 86 imples large alphabet V
        movge r10, #67 // 67 imples large alphabet C

        // MON
        cmp r7, #22
        movge r8, #77 // 77 imples large alphabet M
        movge r9, #79 // 79 imples large alphabet O
        movge r10, #78 // 78 imples large alphabet N

        // ABT
        cmp r7, #23
        movge r8, #65 // 65 imples large alphabet A
        movge r9, #66 // 66 imples large alphabet B
        movge r10, #84 // 84 imples large alphabet T

        // HYP
        cmp r7, #26
        movge r8, #72 // 72 imples large alphabet H
        movge r9, #89 // 89 imples large alphabet Y
        movge r10, #80 // 80 imples large alphabet P

        // UND
        cmp r7, #27
        movge r8, #85 // 85 imples large alphabet U
        movge r9, #78 // 78 imples large alphabet N
        movge r10, #68 // 68 imples large alphabet D

        // SYS
        cmp r7, #31
        movge r8, #83 // 83 imples large alphabet S
        movge r9, #89 // 89 imples large alphabet Y
        movge r10, #83 // 83 imples large alphabet S

        strgeb          r8, [r0, #0x30]  @ fill the TxFIFO with 0x48
        strgeb          r9, [r0, #0x30]  @ fill the TxFIFO with 0x48
        strgeb          r10, [r0, #0x30] @ fill the TxFIFO with 0x48

        ldr r1, =string_current_mode

        loop_7:

        // ---------  Check to see if the Tx FIFO is empty
------------------------------
        ldr     r2, [r0, #0x2C]  @ get Channel Status Register
        and     r2, r2, #0x8            @ get Transmit Buffer Empty bit(bit[3:3])
        cmp     r2, #0x8                               @ check if TxFIFO is empty
and ready to receive new data
        bne     loop_7          @ if TxFIFO is NOT empty, keep checking until it is
empty

//-----------------------------------------------------------------------------
---
```

```
        ldrb      r3, [r1], #1
        strb              r3, [r0, #0x30]  @ fill the TxFIFO with 0x48
        cmp       r3, #0x00
        bne               loop_7

        add sp, sp, #4
        mov r5, #0

        TRANSMIT_loop_final:

        // ---------  Check to see if the Tx FIFO is empty
------------------------------
        ldr      r2, [r0, #0x2C]  @ get Channel Status Register
        and      r2, r2, #0x8             @ get Transmit Buffer Empty bit(bit[3:3])
        cmp      r2, #0x8                              @ check if TxFIFO is empty
and ready to receive new data
        bne      TRANSMIT_loop_final            @ if TxFIFO is NOT empty, keep
checking until it is empty

//----------------------------------------------------------------------------
---

        sub sp, sp, #1
        ldrb      r3, [sp] // register의 상위 2 bit가 출력되기 위함
        mov r6, r3, LSR #4 // 상위 2 bit 중 상위의 bit가 r6에 저장됨
        and r7, r3, #15 // 상위 2 bit 중 하위의 bit가 r7에 저장됨

        cmp r6, #9
        addle r6, r6, #48 // if r6 is less than or equal to 9, then add 48 (based on
ASCII)
        addgt r6, r6, #87 // if r6 is greater than 9, then add 87 (based on ASCII)
        strb              r6, [r0, #0x30]  @ fill the TxFIFO with 0x48

        cmp r7, #9
        addle r7, r7, #48 // if r7 is less than or equal to 9, then add 48 (based on
ASCII)
        addgt r7, r7, #87 // if r7 is greater than 9, then add 87 (based on ASCII)
        strb              r7, [r0, #0x30]  @ fill the TxFIFO with 0x48

        add r5, r5, #1
        cmp r5, #2
        bne               TRANSMIT_loop_final

        ldr r1, =string_1

        ldrb      r3, [r1], #1 // r3 has 8-bit value whereas 4-bit is needed
        strb              r3, [r0, #0x30]  @ fill the TxFIFO with 0x48

        TRANSMIT_loop_final_1:

        // ---------  Check to see if the Tx FIFO is empty
------------------------------
        ldr      r2, [r0, #0x2C]  @ get Channel Status Register
        and      r2, r2, #0x8             @ get Transmit Buffer Empty bit(bit[3:3])
        cmp      r2, #0x8                              @ check if TxFIFO is empty
and ready to receive new data
        bne      TRANSMIT_loop_final_1          @ if TxFIFO is NOT empty, keep
checking until it is empty

//----------------------------------------------------------------------------
---

        sub sp, sp, #1
        ldrb      r3, [sp]
        mov r6, r3, LSR #4
        and r7, r3, #15

        cmp r6, #9
        addle r6, r6, #48 // if r6 is less than or equal to 9, then add 48 (based on
ASCII)
        addgt r6, r6, #87 // if r6 is greater than 9, then add 87 (based on ASCII)
        strb            r6, [r0, #0x30]  @ fill the TxFIFO with 0x48

        cmp r7, #9
        addle r7, r7, #48 // if r7 is less than or equal to 9, then add 48 (based on
ASCII)
        addgt r7, r7, #87 // if r7 is greater than 9, then add 87 (based on ASCII)
        strb            r7, [r0, #0x30]  @ fill the TxFIFO with 0x48

        add r5, r5, #1
        cmp r5, #4
        bne             TRANSMIT_loop_final_1

        ldr r1, =string_final

        loop_8:

        // ---------  Check to see if the Tx FIFO is empty
------------------------------
        ldr     r2, [r0, #0x2C]  @ get Channel Status Register
        and     r2, r2, #0x8            @ get Transmit Buffer Empty bit(bit[3:3])
        cmp     r2, #0x8                                @ check if TxFIFO is empty
and ready to receive new data
        bne     loop_8          @ if TxFIFO is NOT empty, keep checking until it is
empty

//-------------------------------------------------------------------------
---

        ldrb     r3, [r1], #1
        strb            r3, [r0, #0x30]  @ fill the TxFIFO with 0x48
        cmp      r3, #0x00
        bne             loop_8

//      sub sp, sp, #4 // sp points to stack-stored pc register value
//      ldr r4, [sp]
//      add r4, r4, #16

        mov     pc, lr                          @     return to the caller
```

아래는 csd_main.c 코드입니다.

```
int csd_main()
{
	return 0;
}
```