INTRODUCTION TO MICROPROCESSORS | EMBEDDED SYSTEMS DEVELOPMENT

CNG 336

MODULE 2 REPORT

Fatma Erem Aksoy – 2315075

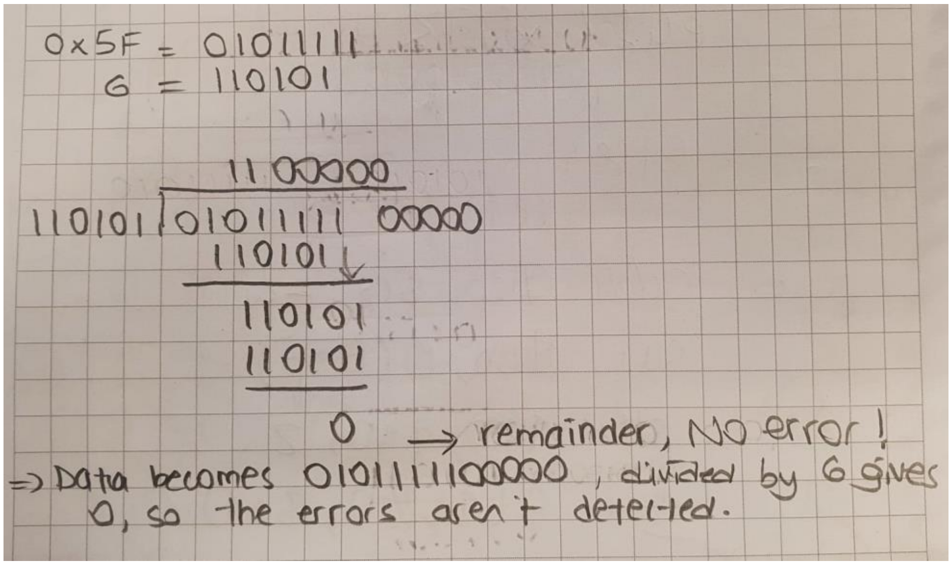
Ece Erseven – 2385383

2.4 DESIGN AND REPORTING

2.4.1 Preliminary Questions

a) **i.** As the packet is 0x00, the remainder will be 0, meaning that there is no CRC error.

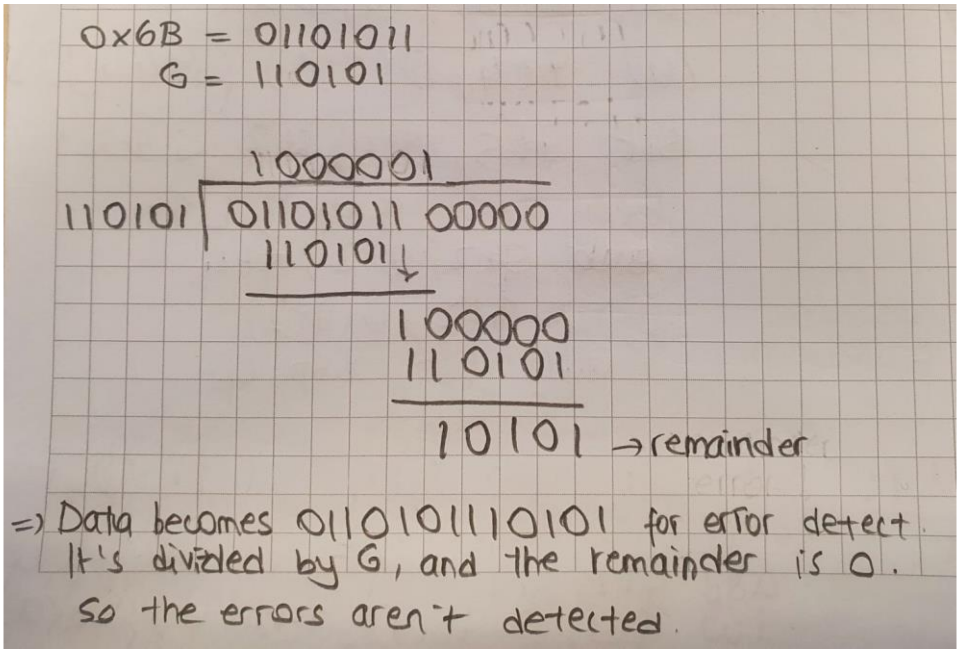
ii. Packet = 0x5F:



```
0x5F = 01011111
   G = 110101

          11000000
110101 | 01011111 00000
          110101
          ------
            110101
            110101
            ------
                 0  → remainder, NO error !
```

⇒ Data becomes 0101111100000, divided by G gives 0, so the errors aren't detected.

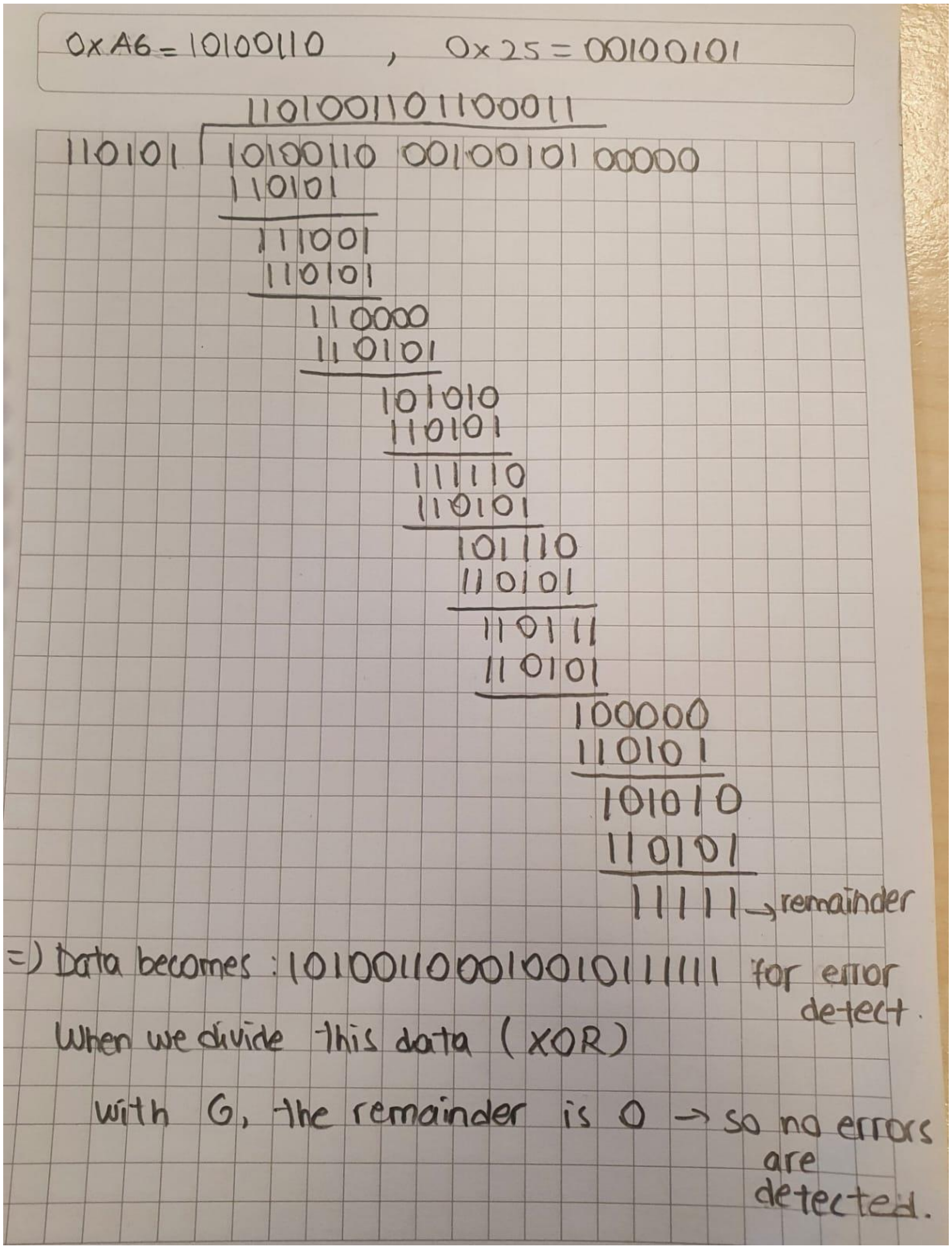
iii. Packet = 0x6B:



```
0x6B = 01101011
   G = 110101

          1000001
110101 | 01101011 00000
          110101
          ------
              100000
              110101
              ------
               10101  → remainder
```

⇒ Data becomes 0110101110101 for error detect. It's divided by G, and the remainder is 0. So the errors aren't detected.

iv. Packet = 0xA6 followed by Packet = 0x25:



0xA6 = 10100110 , 0x25 = 00100101

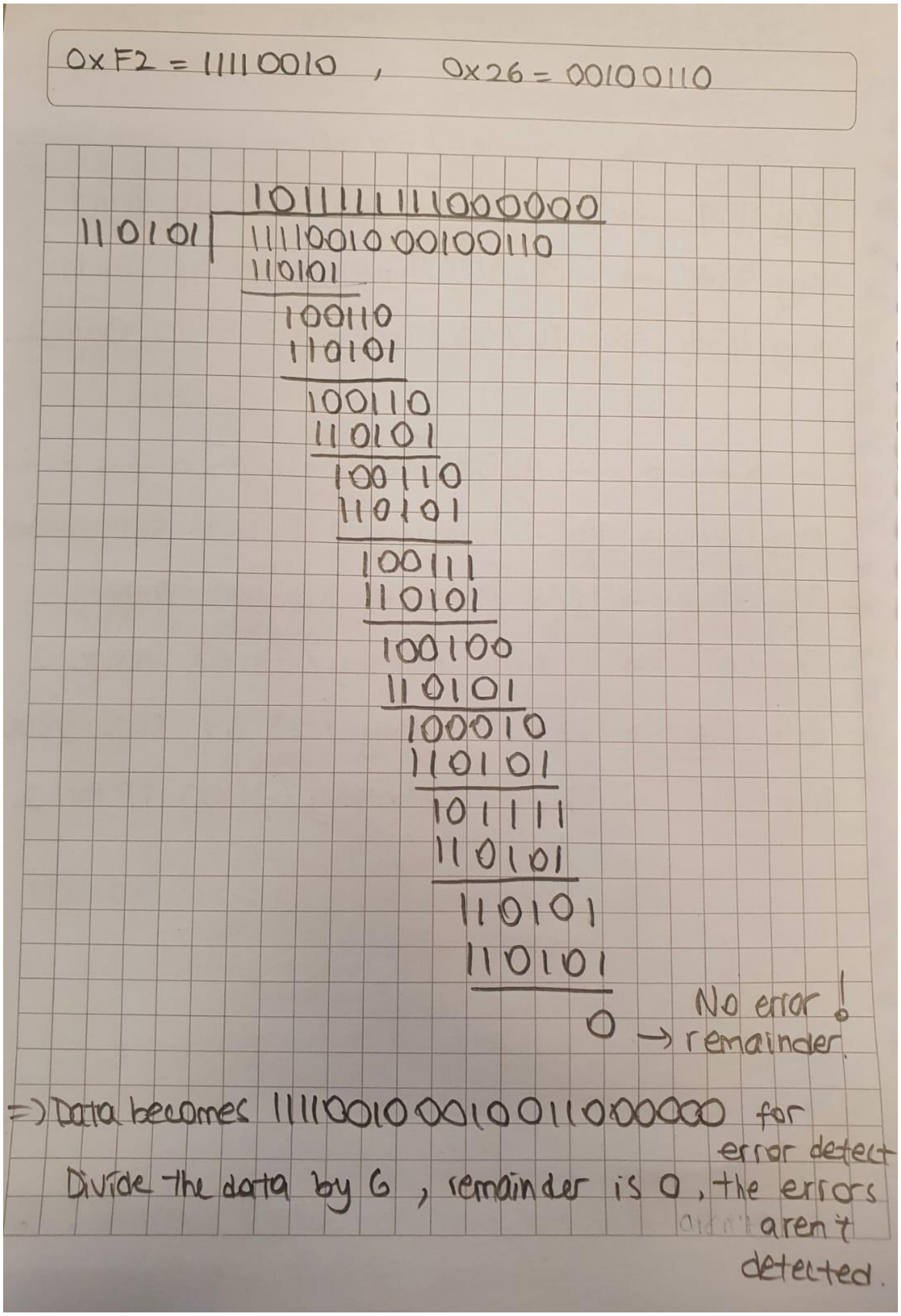
```
                1101001101100011
110101 ) 10100110 00100101 00000
         110101
          111001
          110101
             110000
             110101
                101010
                110101
                 111110
                 110101
                    101110
                    110101
                     110111
                     110101
                          100000
                          110101
                            101010
                            110101
                             11111 → remainder
```

=) Data becomes : 101001100010010111111 for error detect.

When we divide this data (XOR)

with G, the remainder is 0 → so no errors are detected.

v. Packet = 0xF2 followed by Packet = 0x26:



0xF2 = 11110010 , 0x26 = 00100110

```
         1011111111000000
110101 | 1111001000100110
         110101
          100110
          110101
           100110
           110101
            100110
            110101
             100111
             110101
              100100
              110101
               100010
               110101
                101111
                110101
                 110101
                 110101
                      0 → remainder
```

No error !

⇒ Data becomes 1111001000100110000000 for error detect

Divide the data by G , remainder is 0 , the errors aren't detected.

b) **i. ROM:** ROM is a storage unit used in computers and other electronic devices. It is not a storage unit that can be written and erased like RAM. ROM content is written only at the time of production. It cannot be programmed according to the user's own request.

ii. SRAM: SRAM (static RAM) is a type of random access memory (RAM) that retains data bits in a static form, meaning that its memory as long as power is being supplied.

iii. DRAM: Dynamic Random Access Memory is a type of RAM that stores each bit of data in a separate capacitor within an integrated circuit of dynamic random access memory. Since capacitors will discharge after a while due to their nature, they need a refresh/refresh circuit.

iv. SDRAM: SDRAM, or Synchronous Dynamic Random Access Memory is a form of DRAM semiconductor memory can run at faster speeds than conventional DRAM, and it is widely used as the random access memory in a computer, etc. It has high power consumption, and is slower than SRAM.

v. DDR3 SDRAM: It is a further development of the double data rate type of SDRAM. DDR3 was the second generation of double data rate SDRAM and it provided a significant increase in performance and improvement in overall speed.

vi. FLASH: Flash memory is a long-life and non-volatile storage chip that is widely used in embedded systems. It can keep stored data and information even when the power is off. It can be electrically erased and reprogrammed. Flash memory was developed from EEPROM.

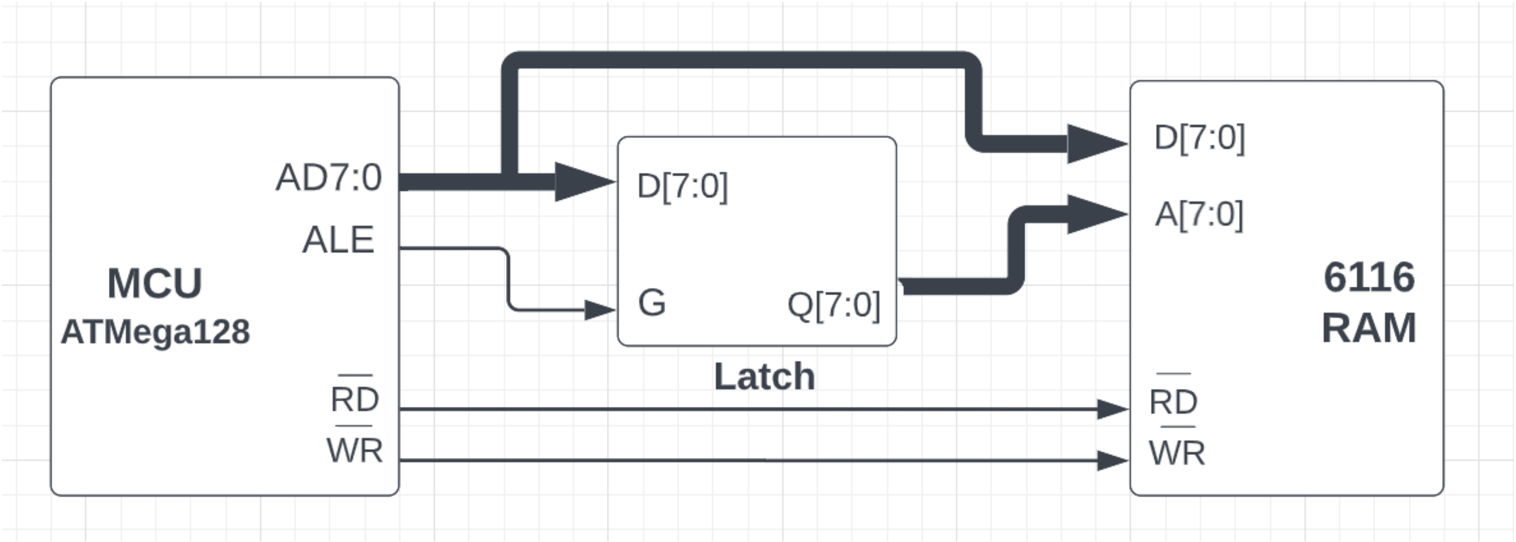
➔ 6116 RAM is a type of SRAM. It is a high-speed SRAM and it is designed as 2Kx8, meaning that it consists of a memory matrix of 2K words of 8-bit each, addressed by 11 address inputs.

c) 8-bit latch-based register is needed in ATMega128 external memory interface design, because it helps to divide the address bits. ATMeaga128 doesn't have separate units for address and data registers, so 8-bit latch provides this feature. It separates them and get the addresses into separate registers to make sure that everything works fine. If we exclude it from the design, SRAM will not understand which bits to take for the address or the data. So it will cause a problem. So it basically does address decoding.

2.4.2 Design

a) External memory interface:

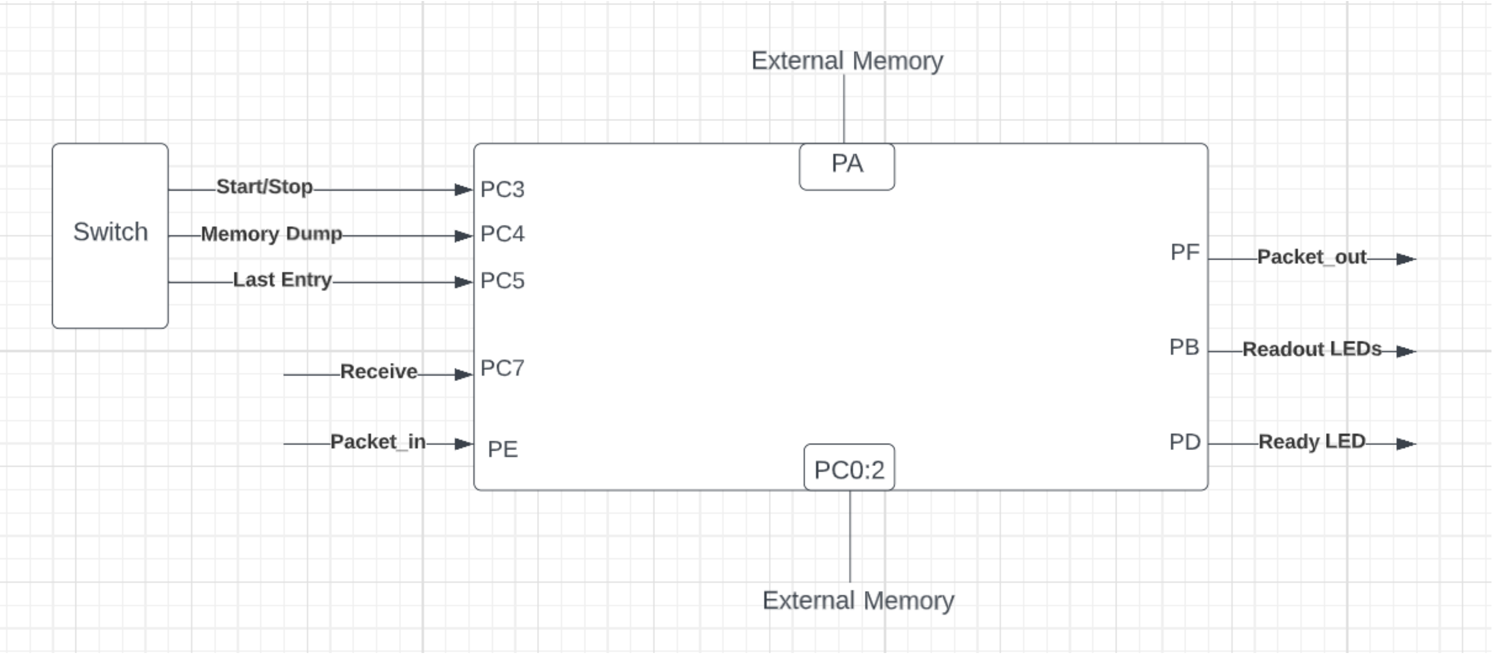
-> No need for address decoding



b)

PORTS (MCU Pins)	**USE**
PORTA	Reserved for the external memory addressing
PORTB	Readout [7:0] output LEDs
PORTC	Receive Button – control pins (pin3 & pin5 & pin7), pin0 and pin2 are reserved for external memory addressing
PORTD	Ready LED
PORTE	Packet-in Input Switches
PORTF	Packet-out LED Output

c) System Layout



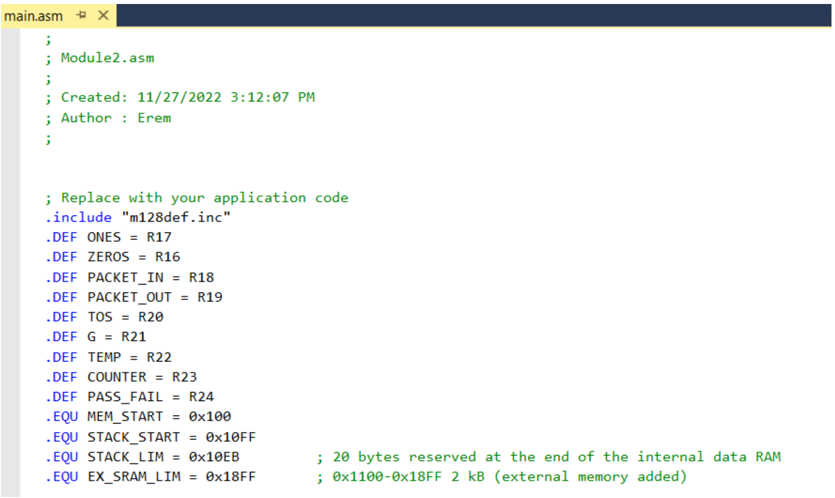
d) The other subroutines added extra are Initialize, STACK_INIT, SERVICE_AND_START, RECEIVE_CHECK, NOT_COMMAND_TYPE, PUSH_PACKET_IN, COMMAND_TYPE, DATA_PACKET, FAIL_CRC_11, PASS_CRC_11, TOS_TO_PACKET_OUT, NO_DATA_PACKET, FAILED, PASSED, REPEAT_CHECK, STACK_CHECK, DATA_LOGGER, EXTEND_MEM_CHECK, WRITE, XRAM, IRAM, PROCEED, CORRECT, LOOP1, LOOP2, CHECK_RESULTING_CRC, CORRECT_11, CHECK_MSB_ONE, RECEIVE, NOT_ASSERTED, MEM_DUMP, CHECK_MEM_EXT, LAST_ENTRY, DELAY, L1, L2, L3.

-> The subroutines that call others are:

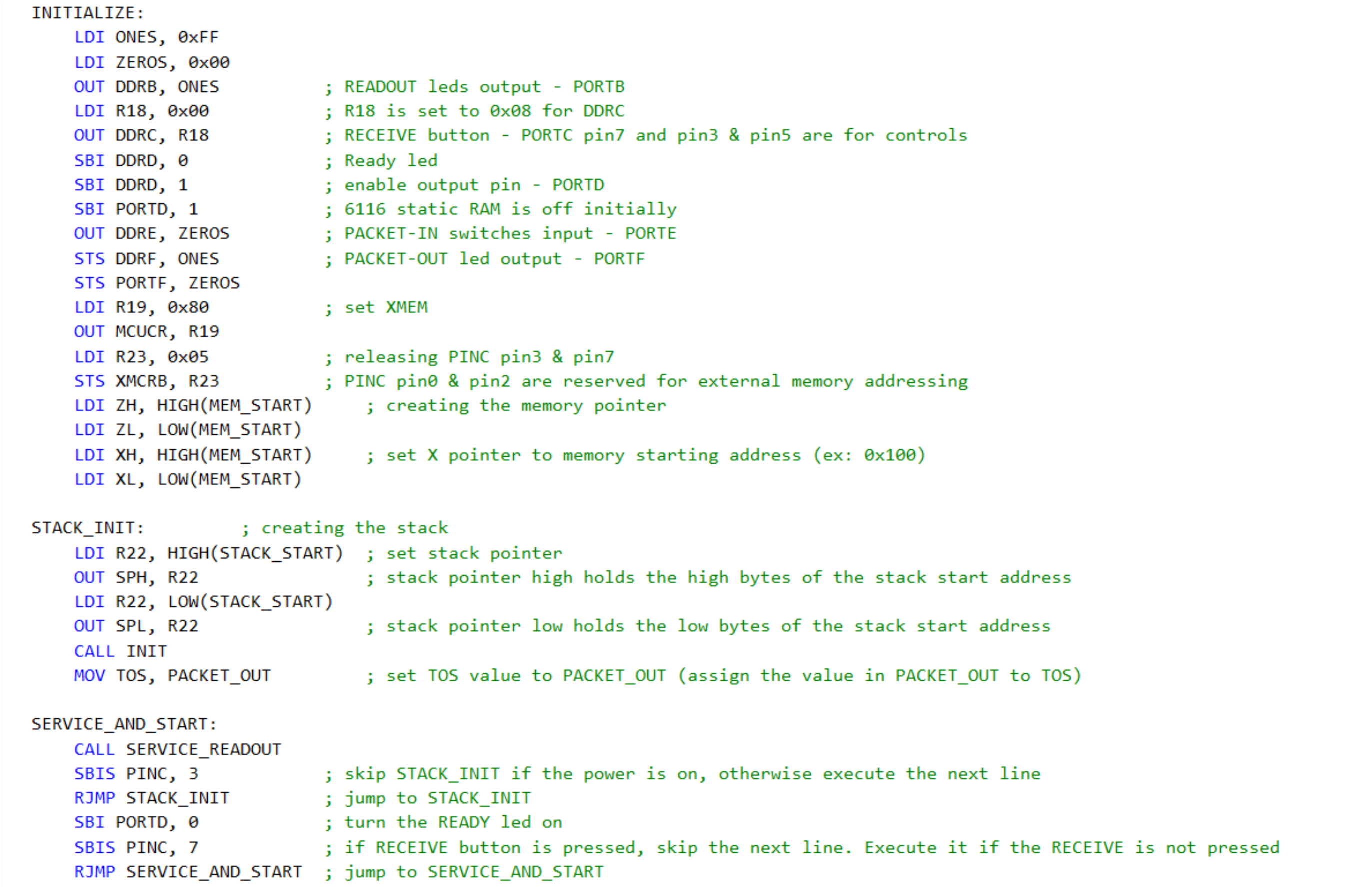
- STACK_INIT is calling INIT
- SERVICE_AND_START is calling STACK_INIT
- RECEIVE_CHECK is calling RECEIVE, NOT_COMMAND_TYPE and COMMAND_TYPE
- NOT_COMMAND_TYPE is calling PUSH_PACKET_IN
- PUSH_PACKET_IN is calling SERVICE_AND_START
- COMMAND_TYPE is calling DATA_PACKET and NO_DATA_PACKET
- DATA_PACKET is calling CRC_CHECK11, FAIL_CRC_11 and PASS_CRC_11
- FAIL_CRC_11 is calling REPEAT_REQUEST
- PASS_CRC_11 is calling INITIALIZE and DATA_LOGGER

- TOS_TO_PACKET_OUT is calling TRANSMIT and SERVICE_AND_START
- NO_DATA_PACKET is calling CRC_CHECK3, FAILED and PASSED
- FAILED is calling REPEAT_REQUEST and SERVICE_AND_START
- PASSED is calling STACK_CHECK and REPEAT_CHECK
- REPEAT_CHECK is calling SERVICE_AND_START and TOS_TO_PACKET_OUT
- STACK_CHECK is calling SERVICE_AND_START
- CRC3 MACRO is calling CHECK_MSB_ONE
- DATA_LOGGER is calling EXTEND_MEM_CHECK and WRITE
- EXTEND_MEM_CHECK is calling WRITE
- WRITE is calling IRAM
- XRAM is calling PROCEED
- PROCEED is calling CRC3
- REPEAT_REQUEST is calling CRC3 and TRANSMIT
- CRC_CHECK3 is calling CRC3 and CORRECT
- LOOP1 is calling CHECK_RESULTING_CRC and CHECK_RESULTING_CRC
- LOOP2 is calling LOOP1
- CHECK_RESULTING_CRC is calling CORRECT_11
- RECEIVE is calling DELAY
- INIT is calling CRC3 and TRANSMIT
- SERVICE_READOUT is calling NOT_ASSERTED, MEM_DUMP and LAST_ENTRY
- NOT_ASSERTED is calling MEM_DUMP and LAST_ENTRY
- MEM_DUMP is calling CHECK_MEM_EXT and DUMP
- CHECK_MEM_EXT is calling DUMP
- DUMP is calling DELAY and MEM_DUMP
- LAST_ENTRY is calling CONTINUE
- DELAY is calling L1, L2 AND L3

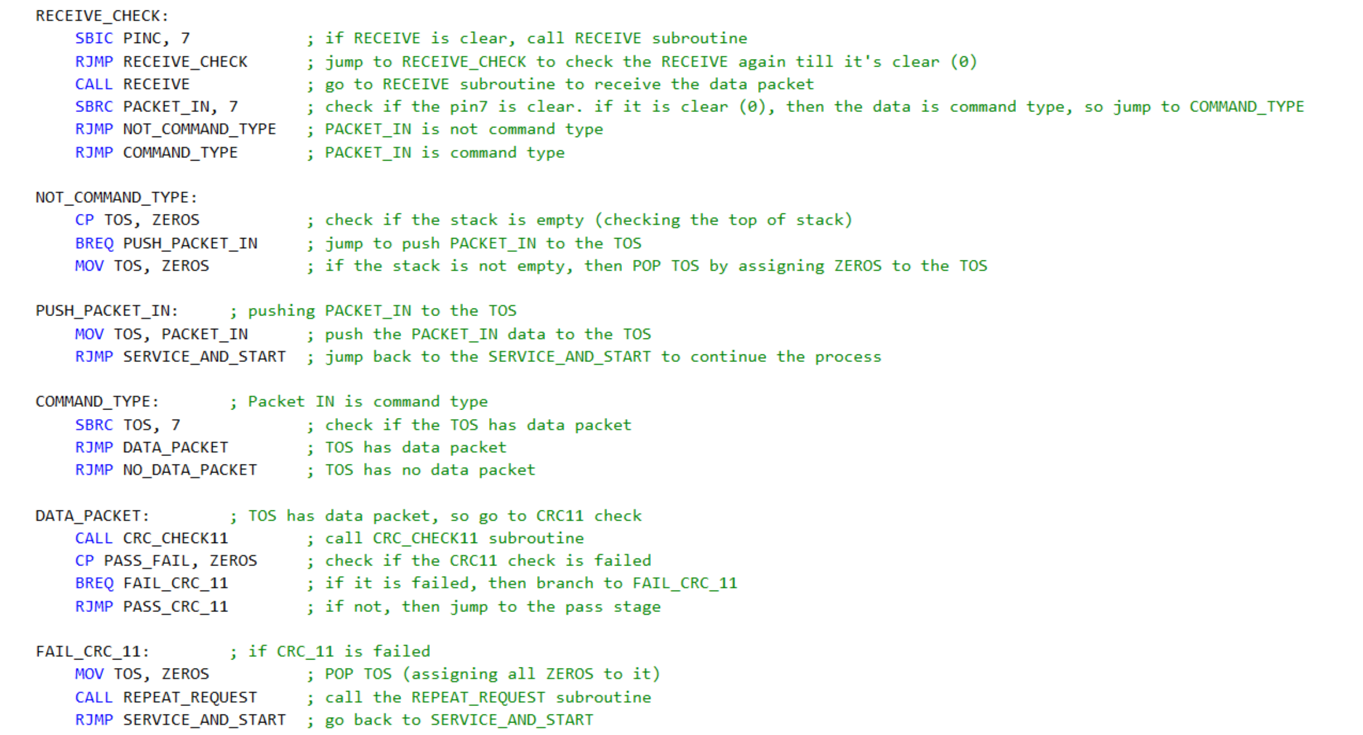
The code:



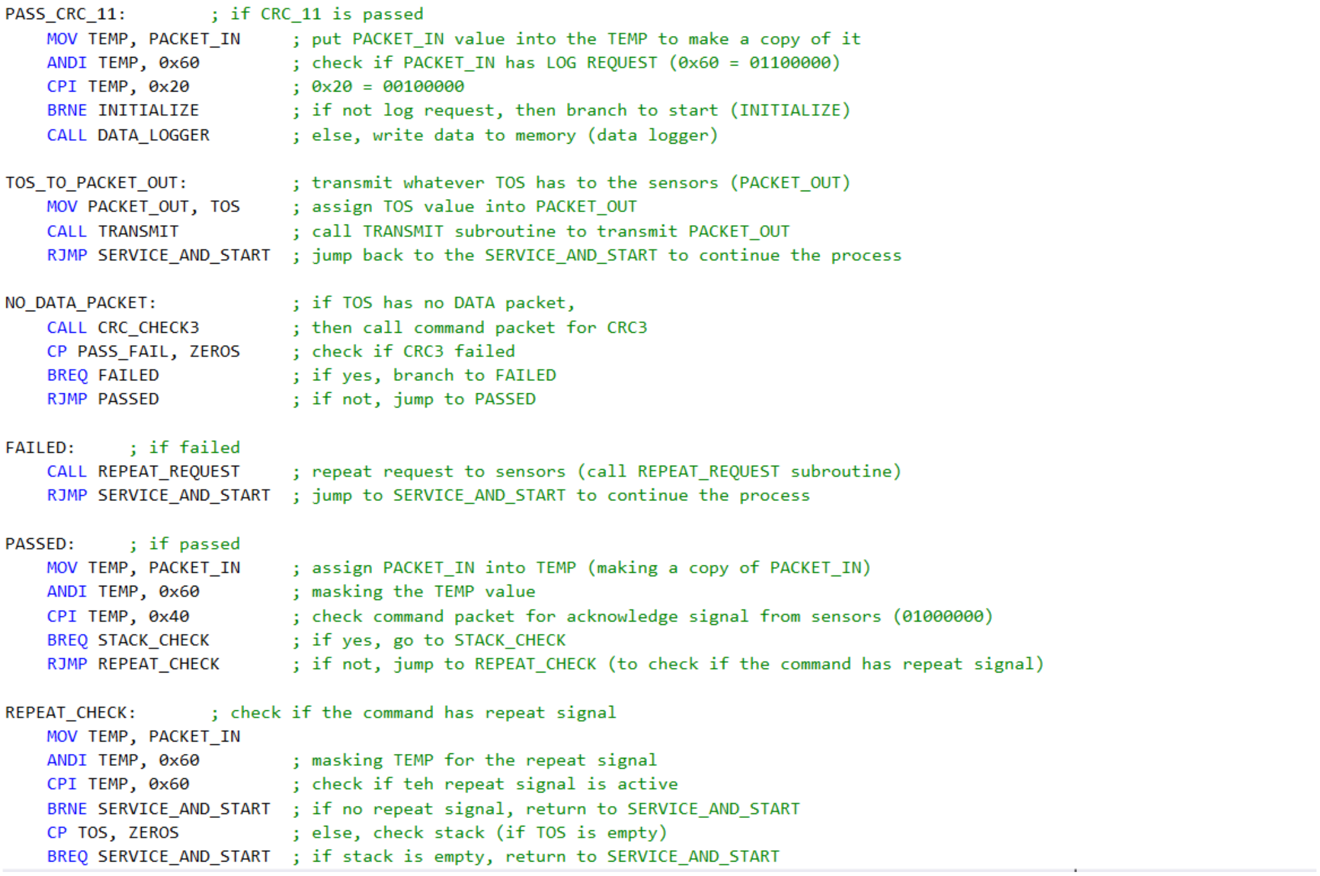
```
;
; Module2.asm
;
; Created: 11/27/2022 3:12:07 PM
; Author : Erem
;


; Replace with your application code
.include "m128def.inc"
.DEF ONES = R17
.DEF ZEROS = R16
.DEF PACKET_IN = R18
.DEF PACKET_OUT = R19
.DEF TOS = R20
.DEF G = R21
.DEF TEMP = R22
.DEF COUNTER = R23
.DEF PASS_FAIL = R24
.EQU MEM_START = 0x100
.EQU STACK_START = 0x10FF
.EQU STACK_LIM = 0x10EB         ; 20 bytes reserved at the end of the internal data RAM
.EQU EX_SRAM_LIM = 0x18FF       ; 0x1100-0x18FF 2 kB (external memory added)
```



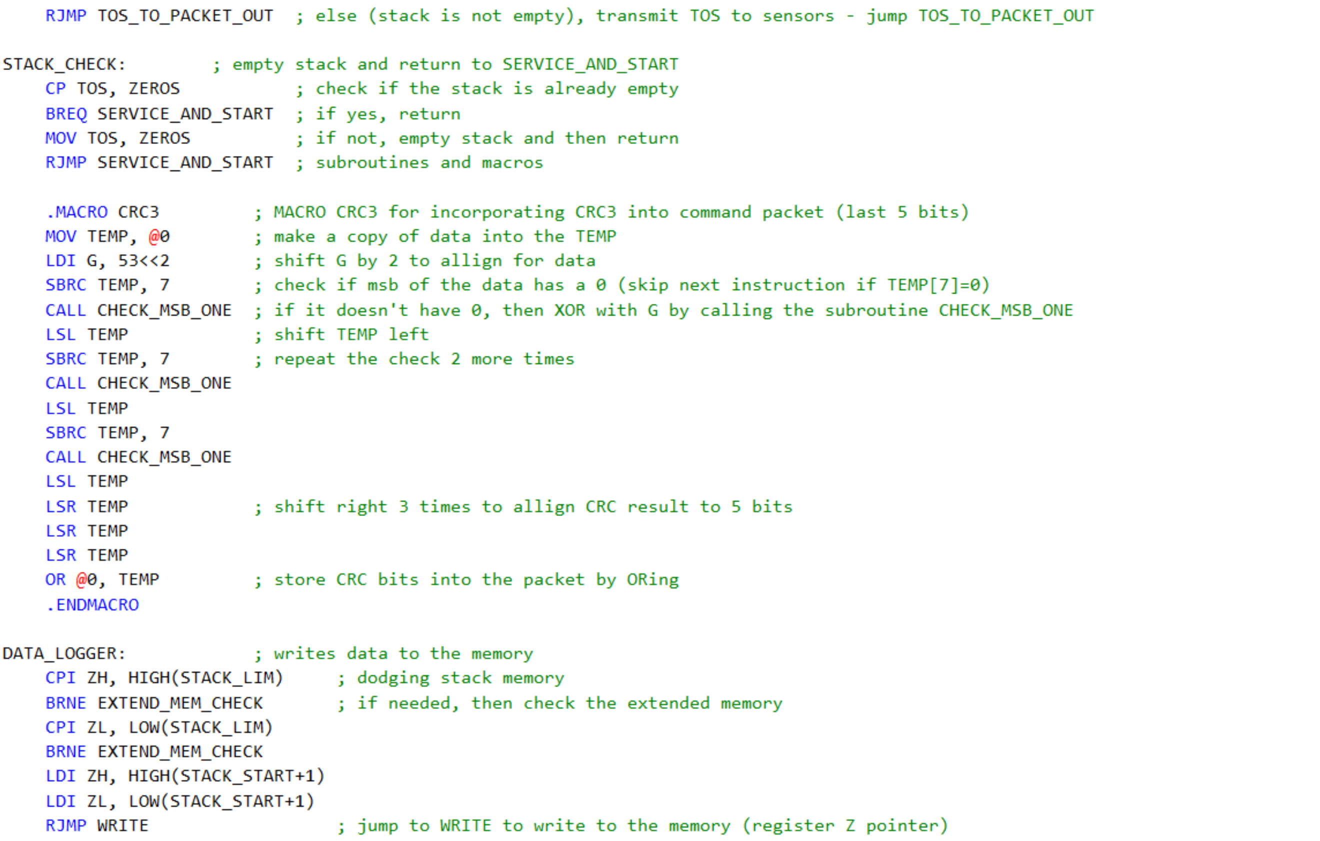
```
INITIALIZE:
    LDI ONES, 0xFF
    LDI ZEROS, 0x00
    OUT DDRB, ONES          ; READOUT leds output - PORTB
    LDI R18, 0x00           ; R18 is set to 0x08 for DDRC
    OUT DDRC, R18           ; RECEIVE button - PORTC pin7 and pin3 & pin5 are for controls
    SBI DDRD, 0             ; Ready led
    SBI DDRD, 1             ; enable output pin - PORTD
    SBI PORTD, 1            ; 6116 static RAM is off initially
    OUT DDRE, ZEROS         ; PACKET-IN switches input - PORTE
    STS DDRF, ONES          ; PACKET-OUT led output - PORTF
    STS PORTF, ZEROS
    LDI R19, 0x80           ; set XMEM
    OUT MCUCR, R19
    LDI R23, 0x05           ; releasing PINC pin3 & pin7
    STS XMCRB, R23          ; PINC pin0 & pin2 are reserved for external memory addressing
    LDI ZH, HIGH(MEM_START)     ; creating the memory pointer
    LDI ZL, LOW(MEM_START)
    LDI XH, HIGH(MEM_START)     ; set X pointer to memory starting address (ex: 0x100)
    LDI XL, LOW(MEM_START)

STACK_INIT:         ; creating the stack
    LDI R22, HIGH(STACK_START)  ; set stack pointer
    OUT SPH, R22                ; stack pointer high holds the high bytes of the stack start address
    LDI R22, LOW(STACK_START)
    OUT SPL, R22                ; stack pointer low holds the low bytes of the stack start address
    CALL INIT
    MOV TOS, PACKET_OUT         ; set TOS value to PACKET_OUT (assign the value in PACKET_OUT to TOS)

SERVICE_AND_START:
    CALL SERVICE_READOUT
    SBIS PINC, 3            ; skip STACK_INIT if the power is on, otherwise execute the next line
    RJMP STACK_INIT         ; jump to STACK_INIT
    SBI PORTD, 0            ; turn the READY led on
    SBIS PINC, 7            ; if RECEIVE button is pressed, skip the next line. Execute it if the RECEIVE is not pressed
    RJMP SERVICE_AND_START  ; jump to SERVICE_AND_START
```



```
RECEIVE_CHECK:
    SBIC PINC, 7            ; if RECEIVE is clear, call RECEIVE subroutine
    RJMP RECEIVE_CHECK      ; jump to RECEIVE_CHECK to check the RECEIVE again till it's clear (0)
    CALL RECEIVE            ; go to RECEIVE subroutine to receive the data packet
    SBRC PACKET_IN, 7       ; check if the pin7 is clear. if it is clear (0), then the data is command type, so jump to COMMAND_TYPE
    RJMP NOT_COMMAND_TYPE   ; PACKET_IN is not command type
    RJMP COMMAND_TYPE       ; PACKET_IN is command type

NOT_COMMAND_TYPE:
    CP TOS, ZEROS           ; check if the stack is empty (checking the top of stack)
    BREQ PUSH_PACKET_IN     ; jump to push PACKET_IN to the TOS
    MOV TOS, ZEROS          ; if the stack is not empty, then POP TOS by assigning ZEROS to the TOS

PUSH_PACKET_IN:     ; pushing PACKET_IN to the TOS
    MOV TOS, PACKET_IN      ; push the PACKET_IN data to the TOS
    RJMP SERVICE_AND_START  ; jump back to the SERVICE_AND_START to continue the process

COMMAND_TYPE:       ; Packet IN is command type
    SBRC TOS, 7             ; check if the TOS has data packet
    RJMP DATA_PACKET        ; TOS has data packet
    RJMP NO_DATA_PACKET     ; TOS has no data packet

DATA_PACKET:        ; TOS has data packet, so go to CRC11 check
    CALL CRC_CHECK11        ; call CRC_CHECK11 subroutine
    CP PASS_FAIL, ZEROS     ; check if the CRC11 check is failed
    BREQ FAIL_CRC_11        ; if it is failed, then branch to FAIL_CRC_11
    RJMP PASS_CRC_11        ; if not, then jump to the pass stage

FAIL_CRC_11:        ; if CRC_11 is failed
    MOV TOS, ZEROS          ; POP TOS (assigning all ZEROS to it)
    CALL REPEAT_REQUEST     ; call the REPEAT_REQUEST subroutine
    RJMP SERVICE_AND_START  ; go back to SERVICE_AND_START
```



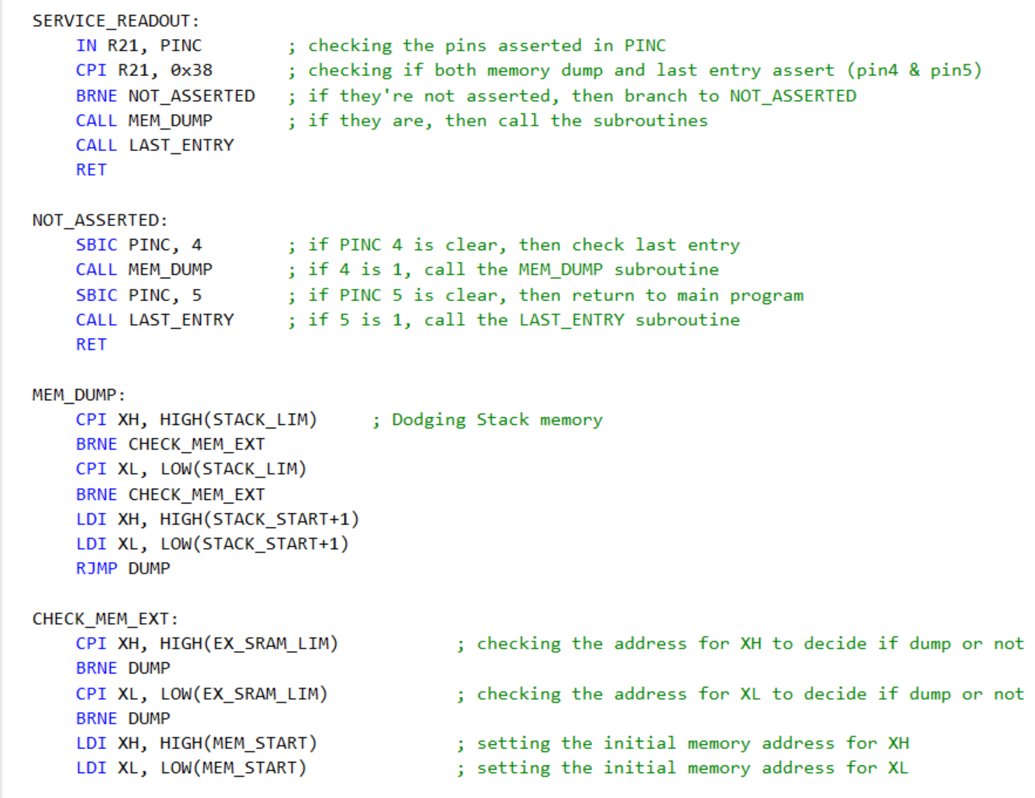
```
PASS_CRC_11:        ; if CRC_11 is passed
    MOV TEMP, PACKET_IN     ; put PACKET_IN value into the TEMP to make a copy of it
    ANDI TEMP, 0x60         ; check if PACKET_IN has LOG REQUEST (0x60 = 01100000)
    CPI TEMP, 0x20          ; 0x20 = 00100000
    BRNE INITIALIZE         ; if not log request, then branch to start (INITIALIZE)
    CALL DATA_LOGGER        ; else, write data to memory (data logger)

TOS_TO_PACKET_OUT:          ; transmit whatever TOS has to the sensors (PACKET_OUT)
    MOV PACKET_OUT, TOS     ; assign TOS value into PACKET_OUT
    CALL TRANSMIT           ; call TRANSMIT subroutine to transmit PACKET_OUT
    RJMP SERVICE_AND_START  ; jump back to the SERVICE_AND_START to continue the process

NO_DATA_PACKET:             ; if TOS has no DATA packet,
    CALL CRC_CHECK3         ; then call command packet for CRC3
    CP PASS_FAIL, ZEROS     ; check if CRC3 failed
    BREQ FAILED             ; if yes, branch to FAILED
    RJMP PASSED             ; if not, jump to PASSED

FAILED:     ; if failed
    CALL REPEAT_REQUEST     ; repeat request to sensors (call REPEAT_REQUEST subroutine)
    RJMP SERVICE_AND_START  ; jump to SERVICE_AND_START to continue the process

PASSED:     ; if passed
    MOV TEMP, PACKET_IN     ; assign PACKET_IN into TEMP (making a copy of PACKET_IN)
    ANDI TEMP, 0x60         ; masking the TEMP value
    CPI TEMP, 0x40          ; check command packet for acknowledge signal from sensors (01000000)
    BREQ STACK_CHECK        ; if yes, go to STACK_CHECK
    RJMP REPEAT_CHECK       ; if not, jump to REPEAT_CHECK (to check if the command has repeat signal)

REPEAT_CHECK:       ; check if the command has repeat signal
    MOV TEMP, PACKET_IN
    ANDI TEMP, 0x60         ; masking TEMP for the repeat signal
    CPI TEMP, 0x60          ; check if teh repeat signal is active
    BRNE SERVICE_AND_START  ; if no repeat signal, return to SERVICE_AND_START
    CP TOS, ZEROS           ; else, check stack (if TOS is empty)
    BREQ SERVICE_AND_START  ; if stack is empty, return to SERVICE_AND_START
```



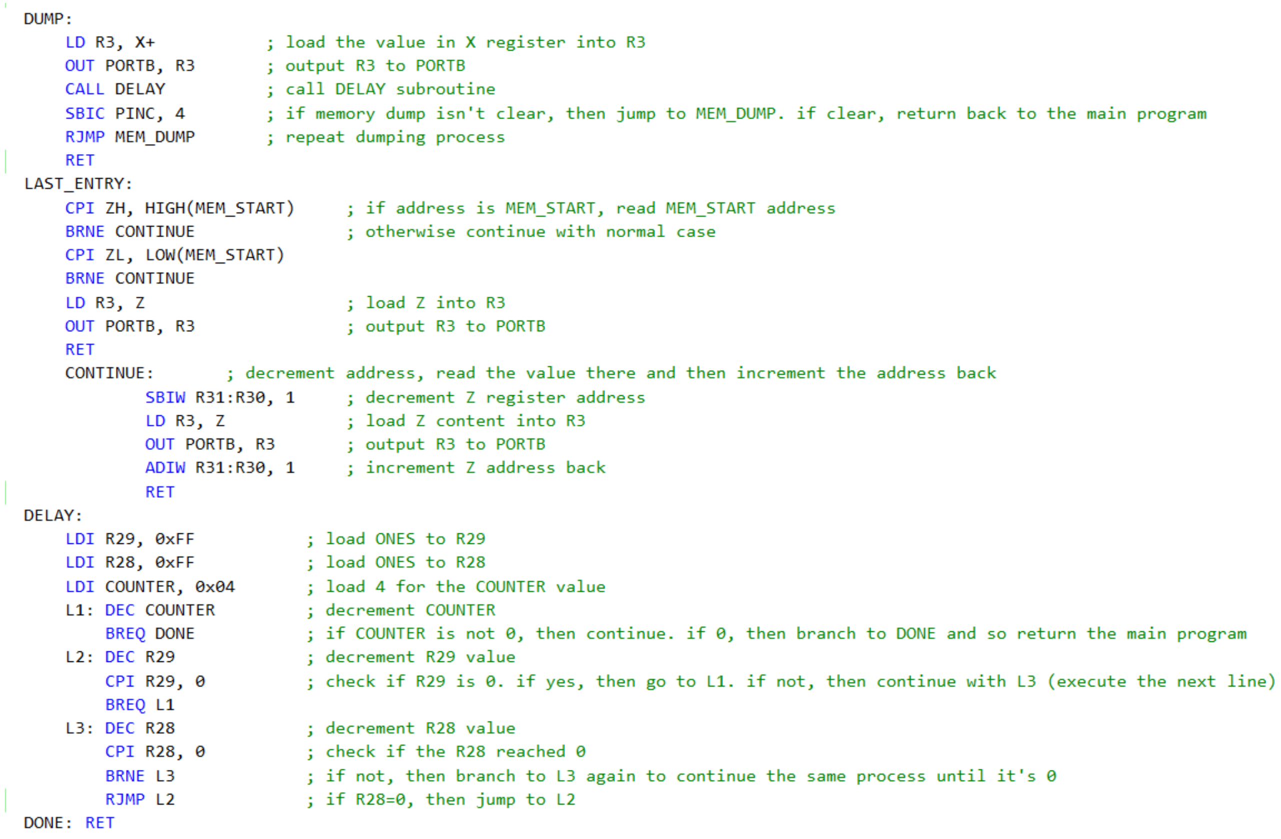
```
    RJMP TOS_TO_PACKET_OUT  ; else (stack is not empty), transmit TOS to sensors - jump TOS_TO_PACKET_OUT

STACK_CHECK:        ; empty stack and return to SERVICE_AND_START
    CP TOS, ZEROS           ; check if the stack is already empty
    BREQ SERVICE_AND_START  ; if yes, return
    MOV TOS, ZEROS          ; if not, empty stack and then return
    RJMP SERVICE_AND_START  ; subroutines and macros

    .MACRO CRC3         ; MACRO CRC3 for incorporating CRC3 into command packet (last 5 bits)
    MOV TEMP, @0        ; make a copy of data into the TEMP
    LDI G, 53<<2        ; shift G by 2 to allign for data
    SBRC TEMP, 7        ; check if msb of the data has a 0 (skip next instruction if TEMP[7]=0)
    CALL CHECK_MSB_ONE  ; if it doesn't have 0, then XOR with G by calling the subroutine CHECK_MSB_ONE
    LSL TEMP            ; shift TEMP left
    SBRC TEMP, 7        ; repeat the check 2 more times
    CALL CHECK_MSB_ONE
    LSL TEMP
    SBRC TEMP, 7
    CALL CHECK_MSB_ONE
    LSL TEMP
    LSR TEMP            ; shift right 3 times to allign CRC result to 5 bits
    LSR TEMP
    LSR TEMP
    OR @0, TEMP         ; store CRC bits into the packet by ORing
    .ENDMACRO

DATA_LOGGER:            ; writes data to the memory
    CPI ZH, HIGH(STACK_LIM)     ; dodging stack memory
    BRNE EXTEND_MEM_CHECK       ; if needed, then check the extended memory
    CPI ZL, LOW(STACK_LIM)
    BRNE EXTEND_MEM_CHECK
    LDI ZH, HIGH(STACK_START+1)
    LDI ZL, LOW(STACK_START+1)
    RJMP WRITE                  ; jump to WRITE to write to the memory (register Z pointer)
```

```
EXTEND_MEM_CHECK:
    CPI ZH, HIGH(EX_SRAM_LIM)           ; check if ZH is within the correct interval
    BRNE WRITE                          ; if not, branch to WRITE
    CPI ZL, LOW(EX_SRAM_LIM)            ; check if ZL is within the correct interval
    BRNE WRITE                          ; if not, branch to WRITE
    LDI ZH, HIGH(MEM_START)             ; setting the initial memory address for ZH
    LDI ZL, LOW(MEM_START)              ; setting the initial memory address for ZL

WRITE:
    CPI ZH,33       ; if ZH is 33 or higher, then XRAM
    BRCS IRAM       ; else, IRAM

XRAM:
    CBI PORTD, 1    ; chip enable (pin1 of PORTD) is set to low (0)
    RJMP PROCEED

IRAM:
    SBI PORTD, 1    ; chip enable (pin1 of PORTD) set to high (1)

PROCEED:
    ST Z+, TOS          ; write data to memory (store whatever value in TOS, into the Z register)
    MOV TOS, ZEROS      ; POP TOS
    LDI TOS, 0x40       ; put acknowledge (01000000) to TOS
    CRC3 TOS            ; create CRC3 code for command
    RET

REPEAT_REQUEST:         ; repeat request with PACKET_OUT
    LDI PACKET_OUT, 0x60    ; 0x60 = 0110 : 01 is log request and 10 is acknowledge to send a new request (repeating the request)
    CRC3 PACKET_OUT         ; calling a CRC3 MACRO with PACKET_OUT
    CALL TRANSMIT           ; call the TRANSMIT subroutine to transmit the PACKET_OUT
    RET

CRC_CHECK3:                 ; checking the last 5 bits of command type packet for the correct CRC3 encoding
    MOV R5, PACKET_IN       ; assigning PACKET_IN value to the R5
    CRC3 PACKET_IN          ; calling the MACRO CRC3 with the PACKET_IN data
    CP R5, PACKET_IN        ; compare if the PACKET_IN and R5 has the same value
    BREQ CORRECT            ; if they're equal, then the result is correct, acknowledge in PACKET_IN
    MOV PASS_FAIL, ZEROS    ; if the CRC3 is false, then set PASS_FAIL to all zeros
    RET

CORRECT:        ; if CRC3 is true, then set PASS_FAIL to all ones
    MOV PASS_FAIL, ONES
    RET

CRC_CHECK11:                ; check data packet on TOS + command packet (first 3 bits) for correct CRC3 bit in command (last 5 bits)
    MOV YH, TOS             ; put the data in TOS into high bytes of Y
    MOV YL, PACKET_IN       ; put command (PACKET_IN) into low bytes of Y
    ANDI YL, 0b11100000     ; masking the first 3 bits of lower Y
    LDI COUNTER, 12         ; loading value for the counter for the number of times to apply the XOR operation
    LDI G, 53<<2            ; shift G 2 bits to allign it with the data

LOOP1:  DEC COUNTER                 ; decrease counter
        BREQ CHECK_RESULTING_CRC    ; if done, check the resulting CRC value with the received CRC
        SBRC YH, 7                  ; check msb of data in YH. if zero, shift left
        RJMP LOOP2                  ; else, jump to LOOP2
        LSL YL                      ; shift left YL
        ROL YH
        RJMP LOOP1                  ; repeat the operation till the counter equals to 0

LOOP2:  EOR YH, G       ; XOR data in YH with G
        LSL YL          ; shift left YL
        ROL YH          ; rotate left YH
        RJMP LOOP1      ; return to LOOP1
```

```
CHECK_RESULTING_CRC:        ; check the resulting CRC value with received one
    MOV TEMP, PACKET_IN     ; make a copy of PACKET_IN, assign it to the TEMP
    LSL TEMP                ; shift TEMP left 3 times
    LSL TEMP
    LSL TEMP
    CP TEMP, YH             ; compare CRC from command with the computed one above
    BREQ CORRECT_11         ; if they equal to each other, branch to CORRECT_11
    MOV PASS_FAIL, ZEROS    ; else, set PASS_FAIL value to all zeros
    RET

CORRECT_11:         ; set PASS_FAIL to all ones
    MOV PASS_FAIL, ONES
    RET

CHECK_MSB_ONE:      ; check if msb of the data is 1 (the data used in CRC3 computation)
    EOR TEMP,G      ; XOR temp with G
    RET

TRANSMIT:
    STS PORTF, PACKET_OUT ; send PACKET_OUT to PORTF
    RET

RECEIVE:
    CBI PORTD, 0        ; turn the READY led off
    IN PACKET_IN, PINE  ; receive data from PORTE
    CALL DELAY          ; call DELAY subroutine
    RET

INIT:
    LDI PACKET_OUT, 0x000   ; reset request
    CRC3 PACKET_OUT         ; calling a CRC3 MACRO with PACKET_OUT
    CALL TRANSMIT
    RET
```



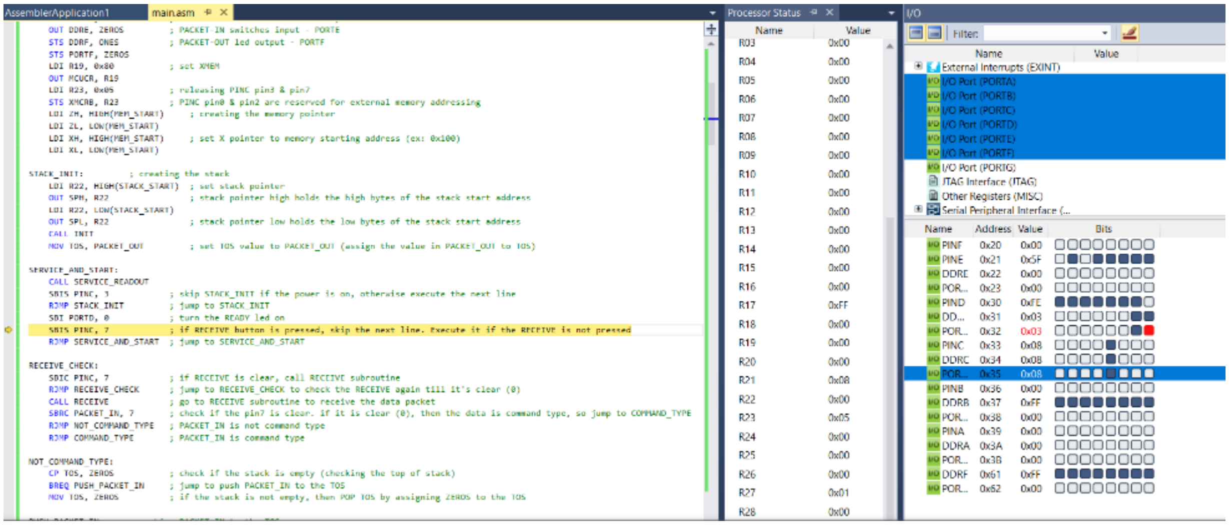
```
SERVICE_READOUT:
    IN R21, PINC        ; checking the pins asserted in PINC
    CPI R21, 0x38       ; checking if both memory dump and last entry assert (pin4 & pin5)
    BRNE NOT_ASSERTED   ; if they're not asserted, then branch to NOT_ASSERTED
    CALL MEM_DUMP       ; if they are, then call the subroutines
    CALL LAST_ENTRY
    RET

NOT_ASSERTED:
    SBIC PINC, 4        ; if PINC 4 is clear, then check last entry
    CALL MEM_DUMP       ; if 4 is 1, call the MEM_DUMP subroutine
    SBIC PINC, 5        ; if PINC 5 is clear, then return to main program
    CALL LAST_ENTRY     ; if 5 is 1, call the LAST_ENTRY subroutine
    RET

MEM_DUMP:
    CPI XH, HIGH(STACK_LIM)     ; Dodging Stack memory
    BRNE CHECK_MEM_EXT
    CPI XL, LOW(STACK_LIM)
    BRNE CHECK_MEM_EXT
    LDI XH, HIGH(STACK_START+1)
    LDI XL, LOW(STACK_START+1)
    RJMP DUMP

CHECK_MEM_EXT:
    CPI XH, HIGH(EX_SRAM_LIM)           ; checking the address for XH to decide if dump or not
    BRNE DUMP
    CPI XL, LOW(EX_SRAM_LIM)            ; checking the address for XL to decide if dump or not
    BRNE DUMP
    LDI XH, HIGH(MEM_START)             ; setting the initial memory address for XH
    LDI XL, LOW(MEM_START)              ; setting the initial memory address for XL
```



```
DUMP:
    LD R3, X+           ; load the value in X register into R3
    OUT PORTB, R3       ; output R3 to PORTB
    CALL DELAY          ; call DELAY subroutine
    SBIC PINC, 4        ; if memory dump isn't clear, then jump to MEM_DUMP. if clear, return back to the main program
    RJMP MEM_DUMP       ; repeat dumping process
    RET
LAST_ENTRY:
    CPI ZH, HIGH(MEM_START)     ; if address is MEM_START, read MEM_START address
    BRNE CONTINUE               ; otherwise continue with normal case
    CPI ZL, LOW(MEM_START)
    BRNE CONTINUE
    LD R3, Z                    ; load Z into R3
    OUT PORTB, R3               ; output R3 to PORTB
    RET
    CONTINUE:       ; decrement address, read the value there and then increment the address back
            SBIW R31:R30, 1     ; decrement Z register address
            LD R3, Z            ; load Z content into R3
            OUT PORTB, R3       ; output R3 to PORTB
            ADIW R31:R30, 1     ; increment Z address back
            RET
DELAY:
    LDI R29, 0xFF           ; load ONES to R29
    LDI R28, 0xFF           ; load ONES to R28
    LDI COUNTER, 0x04       ; load 4 for the COUNTER value
    L1: DEC COUNTER         ; decrement COUNTER
        BREQ DONE           ; if COUNTER is not 0, then continue. if 0, then branch to DONE and so return the main program
    L2: DEC R29             ; decrement R29 value
        CPI R29, 0          ; check if R29 is 0. if yes, then go to L1. if not, then continue with L3 (execute the next line)
        BREQ L1
    L3: DEC R28             ; decrement R28 value
        CPI R28, 0          ; check if the R28 reached 0
        BRNE L3             ; if not, then branch to L3 again to continue the same process until it's 0
        RJMP L2             ; if R28=0, then jump to L2
DONE: RET
```

2.4.3 Verification

-> As a Packet-in value 0x5F is given. In this case, it is found that it is not-command type. After INITIALIZE and STACK_INIT are called, program goes to the INIT and TRANSMIT, respectively. It returns to the SERVICE_AND_START and set the TOS to the PACKET_OUT. Then SERVICE_READOUT is called that checks if the PINC bit 4 and 5 asserted or not. Since in our case just bit 4 is asserted, NOT_ASSERTED is called. It checks bit 4 is cleared or not. If bit4 = 1 it calls MEM_DUMP. But in our case it skips MEM_DUMP call since bit 4 is cleared. Then it checks if bit 5 and since it is also cleared program retuns to the beginning.



AssemblerApplication1 | main.asm

```
    OUT DDRE, ZEROS          ; PACKET-IN switches input - PORTE
    STS DDRF, ONES           ; PACKET-OUT led output - PORTF
    STS PORTF, ZEROS
    LDI R19, 0x80            ; set XMEM
    OUT MCUCR, R19
    LDI R23, 0x05            ; releasing PINC pin3 & pin7
    STS XMCRB, R23           ; PINC pin0 & pin2 are reserved for external memory addressing
    LDI ZH, HIGH(MEM_START)      ; creating the memory pointer
    LDI ZL, LOW(MEM_START)
    LDI XH, HIGH(MEM_START)      ; set X pointer to memory starting address (ex: 0x100)
    LDI XL, LOW(MEM_START)

STACK_INIT:          ; creating the stack
    LDI R22, HIGH(STACK_START)  ; set stack pointer
    OUT SPH, R22                ; stack pointer high holds the high bytes of the stack start address
    LDI R22, LOW(STACK_START)
    OUT SPL, R22                ; stack pointer low holds the low bytes of the stack start address
    CALL INIT
    MOV TOS, PACKET_OUT         ; set TOS value to PACKET_OUT (assign the value in PACKET_OUT to TOS)

SERVICE_AND_START:
    CALL SERVICE_READOUT
    SBIS PINC, 3             ; skip STACK_INIT if the power is on, otherwise execute the next line
    RJMP STACK_INIT          ; jump to STACK_INIT
    SBI PORTD, 0             ; turn the READY led on
    SBIS PINC, 7             ; if RECEIVE button is pressed, skip the next line. Execute it if the RECEIVE is not pressed
    RJMP SERVICE_AND_START   ; jump to SERVICE_AND_START

RECEIVE_CHECK:
    SBIC PINC, 7             ; if RECEIVE is clear, call RECEIVE subroutine
    RJMP RECEIVE_CHECK       ; jump to RECEIVE_CHECK to check the RECEIVE again till it's clear (0)
    CALL RECEIVE             ; go to RECEIVE subroutine to receive the data packet
    SBRC PACKET_IN, 7        ; check if the pin7 is clear. if it is clear (0), then the data is command type, so jump to COMMAND_TYPE
    RJMP NOT_COMMAND_TYPE    ; PACKET_IN is not command type
    RJMP COMMAND_TYPE        ; PACKET_IN is command type

NOT_COMMAND_TYPE:
    CP TOS, ZEROS            ; check if the stack is empty (checking the top of stack)
    BREQ PUSH_PACKET_IN      ; jump to push PACKET_IN to the TOS
    MOV TOS, ZEROS           ; if the stack is not empty, then POP TOS by assigning ZEROS to the TOS
```

Processor Status

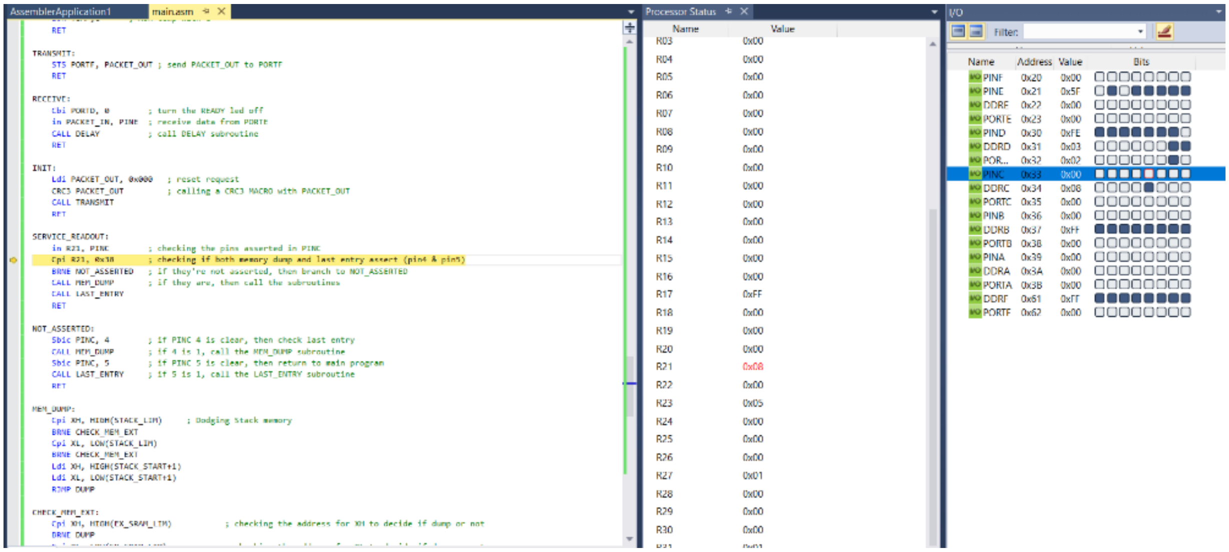
Name	Value
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x00
R17	0xFF
R18	0x00
R19	0x00
R20	0x00
R21	0x08
R22	0x00
R23	0x05
R24	0x00
R25	0x00
R26	0x00
R27	0x01
R28	0x00

I/O

Filter:

Name	Value
External Interrupts (EXINT)	
I/O Port (PORTA)	
I/O Port (PORTB)	
I/O Port (PORTC)	
I/O Port (PORTD)	
I/O Port (PORTE)	
I/O Port (PORTF)	
I/O Port (PORTG)	
JTAG Interface (JTAG)	
Other Registers (MISC)	
Serial Peripheral Interface (...	

Name	Address	Value	Bits
PINF	0x20	0x00	□□□□□□□□
PINE	0x21	0x5F	□■□■■■■■
DDRE	0x22	0x00	□□□□□□□□
POR...	0x23	0x00	□□□□□□□□
PIND	0x30	0xFE	■■■■■■■□
DD...	0x31	0x03	□□□□□□■■
POR...	0x32	0x03	□□□□□□■🟥
PINC	0x33	0x08	□□□□■□□□
DDRC	0x34	0x08	□□□□■□□□
POR...	0x35	0x08	□□□□■□□□
PINB	0x36	0x00	□□□□□□□□
DDRB	0x37	0xFF	■■■■■■■■
POR...	0x38	0x00	□□□□□□□□
PINA	0x39	0x00	□□□□□□□□
DDRA	0x3A	0x00	□□□□□□□□
POR...	0x3B	0x00	□□□□□□□□
DDRF	0x61	0xFF	■■■■■■■■
POR...	0x62	0x00	□□□□□□□□



AssemblerApplication1 | main.asm

```
    RET

TRANSMIT:
    STS PORTF, PACKET_OUT ; send PACKET_OUT to PORTF
    RET

RECEIVE:
    Cbi PORTD, 0          ; turn the READY led off
    in PACKET_IN, PINE    ; receive data from PORTE
    CALL DELAY            ; call DELAY subroutine
    RET

INIT:
    Ldi PACKET_OUT, 0x000   ; reset request
    CRC3 PACKET_OUT         ; calling a CRC3 MACRO with PACKET_OUT
    CALL TRANSMIT
    RET

SERVICE_READOUT:
    in R21, PINC          ; checking the pins asserted in PINC
    Cpi R21, 0x30         ; checking if both memory dump and last entry assert (pin4 & pin5)
    BRNE NOT_ASSERTED     ; If they're not asserted, then branch to NOT_ASSERTED
    CALL MEM_DUMP         ; if they are, then call the subroutines
    CALL LAST_ENTRY
    RET

NOT_ASSERTED:
    Sbic PINC, 4          ; if PINC 4 is clear, then check last entry
    CALL MEM_DUMP         ; if 4 is 1, call the MEM_DUMP subroutine
    Sbic PINC, 5          ; if PINC 5 is clear, then return to main program
    CALL LAST_ENTRY       ; if 5 is 1, call the LAST_ENTRY subroutine
    RET

MEM_DUMP:
    Cpi XH, HIGH(STACK_LIM)     ; Dodging Stack memory
    BRNE CHECK_MEM_EXT
    Cpi XL, LOW(STACK_LIM)
    BRNE CHECK_MEM_EXT
    Ldi XH, HIGH(STACK_START+1)
    Ldi XL, LOW(STACK_START+1)
    RJMP DUMP

CHECK_MEM_EXT:
    Cpi XH, HIGH(EX_SRAM_LIM)          ; checking the address for XH to decide if dump or not
    BRNE DUMP
```

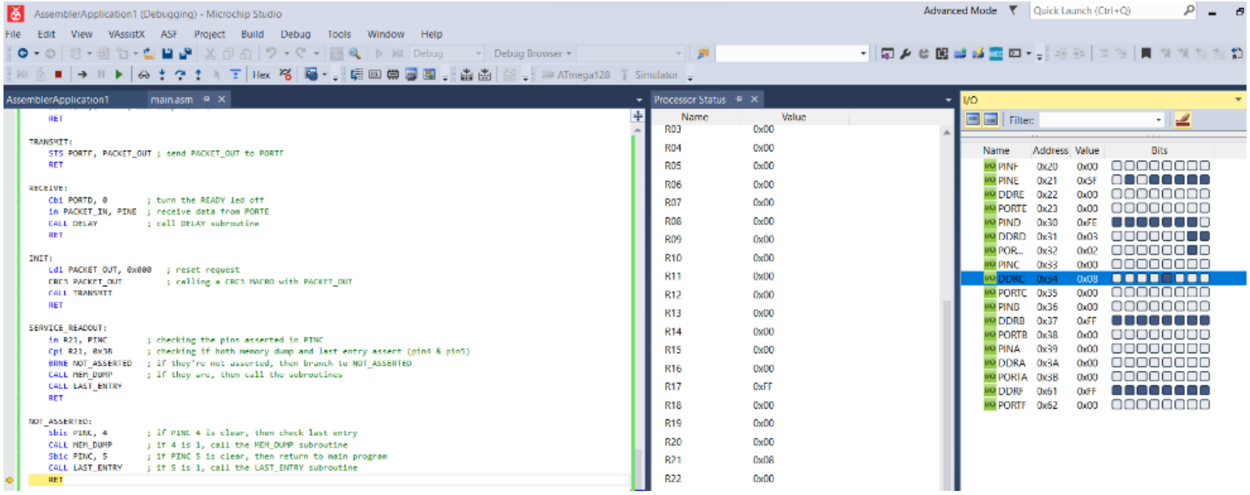
Processor Status

Name	Value
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x00
R17	0xFF
R18	0x00
R19	0x00
R20	0x00
R21	0x08
R22	0x00
R23	0x05
R24	0x00
R25	0x00
R26	0x00
R27	0x01
R28	0x00
R29	0x00
R30	0x00

I/O

Filter:

Name	Address	Value	Bits
PINF	0x20	0x00	□□□□□□□□
PINE	0x21	0x5F	□■□■■■■■
DDRE	0x22	0x00	□□□□□□□□
PORTE	0x23	0x00	□□□□□□□□
PIND	0x30	0xFE	■■■■■■■□
DDRD	0x31	0x03	□□□□□□■■
POR...	0x32	0x02	□□□□□□■□
PINC	0x33	0x00	□□□□□□□□
DDRC	0x34	0x08	□□□□■□□□
PORTC	0x35	0x00	□□□□□□□□
PINB	0x36	0x00	□□□□□□□□
DDRB	0x37	0xFF	■■■■■■■■
PORTB	0x38	0x00	□□□□□□□□
PINA	0x39	0x00	□□□□□□□□
DDRA	0x3A	0x00	□□□□□□□□
PORTA	0x3B	0x00	□□□□□□□□
DDRF	0x61	0xFF	■■■■■■■■
PORTF	0x62	0x00	□□□□□□□□



AssemblerApplication1 (Debugging) - Microchip Studio

AssemblerApplication1 | main.asm

```
    RET

TRANSMIT:
    STS PORTF, PACKET_OUT ; send PACKET_OUT to PORTF
    RET

RECEIVE:
    Cbi PORTD, 0          ; turn the READY led off
    in PACKET_IN, PINE    ; receive data from PORTE
    CALL DELAY            ; call DELAY subroutine
    RET

INIT:
    Ldi PACKET_OUT, 0x000   ; reset request
    CRC3 PACKET_OUT         ; calling a CRC3 MACRO with PACKET_OUT
    CALL TRANSMIT
    RET

SERVICE_READOUT:
    in R21, PINC          ; checking the pins asserted in PINC
    Cpi R21, 0x30         ; checking if both memory dump and last entry assert (pin4 & pin5)
    BRNE NOT_ASSERTED     ; If they're not asserted, then branch to NOT_ASSERTED
    CALL MEM_DUMP         ; if they are, then call the subroutines
    CALL LAST_ENTRY
    RET

NOT_ASSERTED:
    Sbic PINC, 4          ; if PINC 4 is clear, then check last entry
    CALL MEM_DUMP         ; if 4 is 1, call the MEM_DUMP subroutine
    Sbic PINC, 5          ; if PINC 5 is clear, then return to main program
    CALL LAST_ENTRY       ; if 5 is 1, call the LAST_ENTRY subroutine
    RET
```

Processor Status

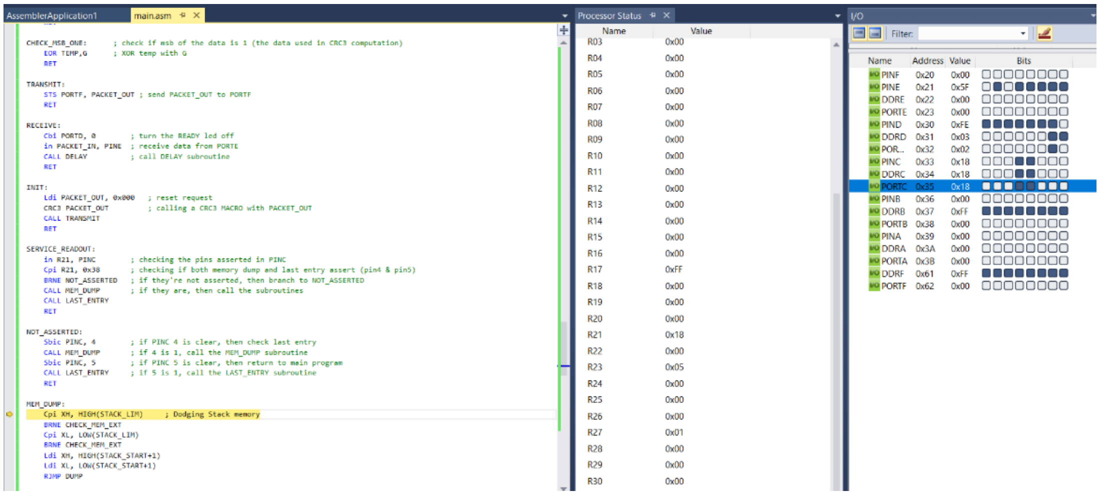
Name	Value
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x00
R17	0xFF
R18	0x00
R19	0x00
R20	0x00
R21	0x08
R22	0x00

I/O

Filter:

Name	Address	Value	Bits
PINF	0x20	0x00	□□□□□□□□
PINE	0x21	0x5F	□■□■■■■■
DDRE	0x22	0x00	□□□□□□□□
PORTE	0x23	0x00	□□□□□□□□
PIND	0x30	0xFE	■■■■■■■□
DDRD	0x31	0x03	□□□□□□■■
POR...	0x32	0x02	□□□□□□■□
PINC	0x33	0x00	□□□□□□□□
DDRC	0x34	0x08	□□□□■□□□
PORTC	0x35	0x00	□□□□□□□□
PINB	0x36	0x00	□□□□□□□□
DDRB	0x37	0xFF	■■■■■■■■
PORTB	0x38	0x00	□□□□□□□□
PINA	0x39	0x00	□□□□□□□□
DDRA	0x3A	0x00	□□□□□□□□
PORTA	0x3B	0x00	□□□□□□□□
DDRF	0x61	0xFF	■■■■■■■■
PORTF	0x62	0x00	□□□□□□□□

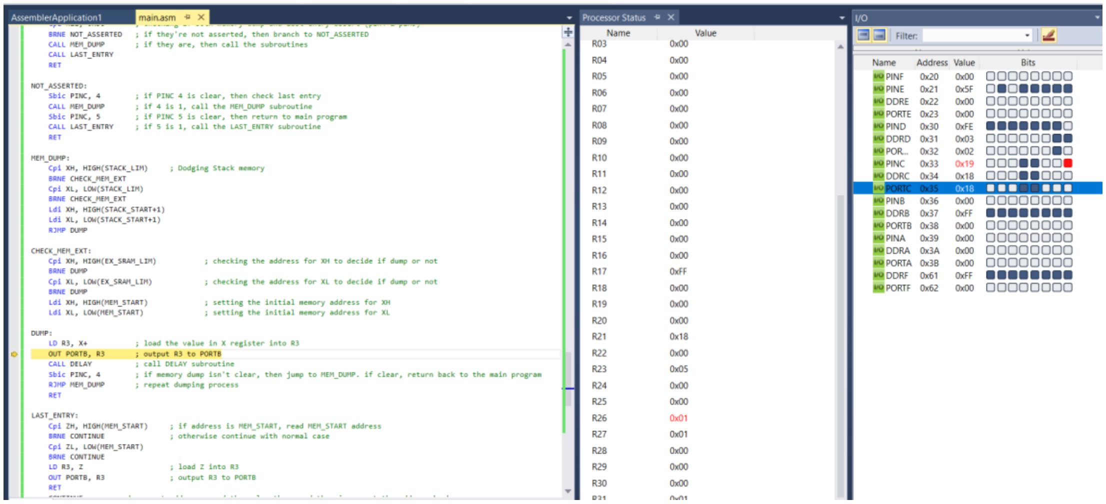
-> Debug 2: We tested the same PACKET_IN value with different control input for this case. Since bit5 of PINC is not asserted it calls the NOT_ASSERTED, but this time bit 4 = 1 so MEM_DUMP is called. Program jumped into DUMP after that and load the value of X register into the R3 and set PORTB to R3, then it called the DELAY.



```
CHECK_MSB_ONE:      ; check if msb of the data is 1 (the data used in CRC3 computation)
    EOR TEMP,G      ; XOR temp with G
    RET

TRANSMIT:
    STS PORTF, PACKET_OUT ; send PACKET_OUT to PORTF
    RET

RECEIVE:
    Cbi PORTD, 0         ; turn the READY led off
    in PACKET_IN, PINE   ; receive data from PORTE
    CALL DELAY           ; call DELAY subroutine
    RET

INIT:
    Ldi PACKET_OUT, 0x000   ; reset request
    CRC3 PACKET_OUT         ; calling a CRC3 MACRO with PACKET_OUT
    CALL TRANSMIT
    RET

SERVICE_READOUT:
    in R21, PINC         ; checking the pins asserted in PINC
    Cpi R21, 0x38        ; checking if both memory dump and last entry assert (pin4 & pin5)
    BRNE NOT_ASSERTED    ; if they're not asserted, then branch to NOT_ASSERTED
    CALL MEM_DUMP        ; if they are, then call the subroutines
    CALL LAST_ENTRY
    RET

NOT_ASSERTED:
    Sbic PINC, 4         ; if PINC 4 is clear, then check last entry
    CALL MEM_DUMP        ; if 4 is 1, call the MEM_DUMP subroutine
    Sbic PINC, 5         ; if PINC 5 is clear, then return to main program
    CALL LAST_ENTRY      ; if 5 is 1, call the LAST_ENTRY subroutine
    RET

MEM_DUMP:
    Cpi XH, HIGH(STACK_LIM)     ; Dodging Stack memory
    BRNE CHECK_MEM_EXT
    Cpi XL, LOW(STACK_LIM)
    BRNE CHECK_MEM_EXT
    Ldi XH, HIGH(STACK_START+1)
    Ldi XL, LOW(STACK_START+1)
    RJMP DUMP
```

Name	Value
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x00
R17	0xFF
R18	0x00
R19	0x00
R20	0x00
R21	0x18
R22	0x00
R23	0x05
R24	0x00
R25	0x00
R26	0x00
R27	0x01
R28	0x00
R29	0x00
R30	0x00

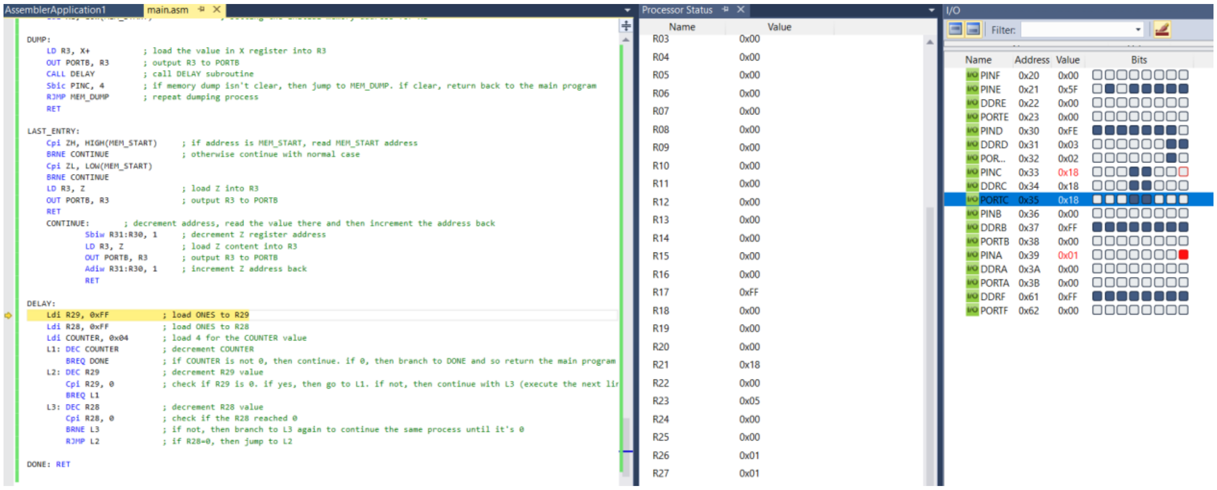
Name	Address	Value	Bits
PINF	0x20	0x00	□□□□□□□□
PINE	0x21	0x5F	□■□■■■■■
DDRE	0x22	0x00	□□□□□□□□
PORTE	0x23	0x00	□□□□□□□□
PIND	0x30	0xFE	■■■■■■■□
DDRD	0x31	0x03	□□□□□□■■
POR...	0x32	0x02	□□□□□□■□
PINC	0x33	0x18	□□□■■□□□
DDRC	0x34	0x18	□□□■■□□□
PORTC	0x35	0x18	□□□■■□□□
PINB	0x36	0x00	□□□□□□□□
DDRB	0x37	0xFF	■■■■■■■■
PORTB	0x38	0x00	□□□□□□□□
PINA	0x39	0x00	□□□□□□□□
DDRA	0x3A	0x00	□□□□□□□□
PORTA	0x3B	0x00	□□□□□□□□
DDRF	0x61	0xFF	■■■■■■■■
PORTF	0x62	0x00	□□□□□□□□



```
    BRNE NOT_ASSERTED    ; if they're not asserted, then branch to NOT_ASSERTED
    CALL MEM_DUMP        ; if they are, then call the subroutines
    CALL LAST_ENTRY
    RET

NOT_ASSERTED:
    Sbic PINC, 4         ; if PINC 4 is clear, then check last entry
    CALL MEM_DUMP        ; if 4 is 1, call the MEM_DUMP subroutine
    Sbic PINC, 5         ; if PINC 5 is clear, then return to main program
    CALL LAST_ENTRY      ; if 5 is 1, call the LAST_ENTRY subroutine
    RET

MEM_DUMP:
    Cpi XH, HIGH(STACK_LIM)     ; Dodging Stack memory
    BRNE CHECK_MEM_EXT
    Cpi XL, LOW(STACK_LIM)
    BRNE CHECK_MEM_EXT
    Ldi XH, HIGH(STACK_START+1)
    Ldi XL, LOW(STACK_START+1)
    RJMP DUMP

CHECK_MEM_EXT:
    Cpi XH, HIGH(EX_SRAM_LIM)           ; checking the address for XH to decide if dump or not
    BRNE DUMP
    Cpi XL, LOW(EX_SRAM_LIM)            ; checking the address for XL to decide if dump or not
    BRNE DUMP
    Ldi XH, HIGH(MEM_START)             ; setting the initial memory address for XH
    Ldi XL, LOW(MEM_START)              ; setting the initial memory address for XL

DUMP:
    LD R3, X+            ; load the value in X register into R3
    OUT PORTB, R3        ; output R3 to PORTB
    CALL DELAY           ; call DELAY subroutine
    Sbic PINC, 4         ; if memory dump isn't clear, then jump to MEM_DUMP. if clear, return back to the main program
    RJMP MEM_DUMP        ; repeat dumping process
    RET

LAST_ENTRY:
    Cpi ZH, HIGH(MEM_START)     ; if address is MEM_START, read MEM_START address
    BRNE CONTINUE               ; otherwise continue with normal case
    Cpi ZL, LOW(MEM_START)
    BRNE CONTINUE
    LD R3, Z                    ; load Z into R3
    OUT PORTB, R3               ; output R3 to PORTB
    RET
```

Name	Value
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x00
R17	0xFF
R18	0x00
R19	0x00
R20	0x00
R21	0x18
R22	0x00
R23	0x05
R24	0x00
R25	0x00
R26	0x01
R27	0x01
R28	0x00
R29	0x00
R30	0x00
R31	0x01

Name	Address	Value	Bits
PINF	0x20	0x00	□□□□□□□□
PINE	0x21	0x5F	□■□■■■■■
DDRE	0x22	0x00	□□□□□□□□
PORTE	0x23	0x00	□□□□□□□□
PIND	0x30	0xFE	■■■■■■■□
DDRD	0x31	0x03	□□□□□□■■
POR...	0x32	0x02	□□□□□□■□
PINC	0x33	0x19	□□□■■□□■
DDRC	0x34	0x18	□□□■■□□□
PORTC	0x35	0x18	□□□■■□□□
PINB	0x36	0x00	□□□□□□□□
DDRB	0x37	0xFF	■■■■■■■■
PORTB	0x38	0x00	□□□□□□□□
PINA	0x39	0x00	□□□□□□□□
DDRA	0x3A	0x00	□□□□□□□□
PORTA	0x3B	0x00	□□□□□□□□
DDRF	0x61	0xFF	■■■■■■■■
PORTF	0x62	0x00	□□□□□□□□



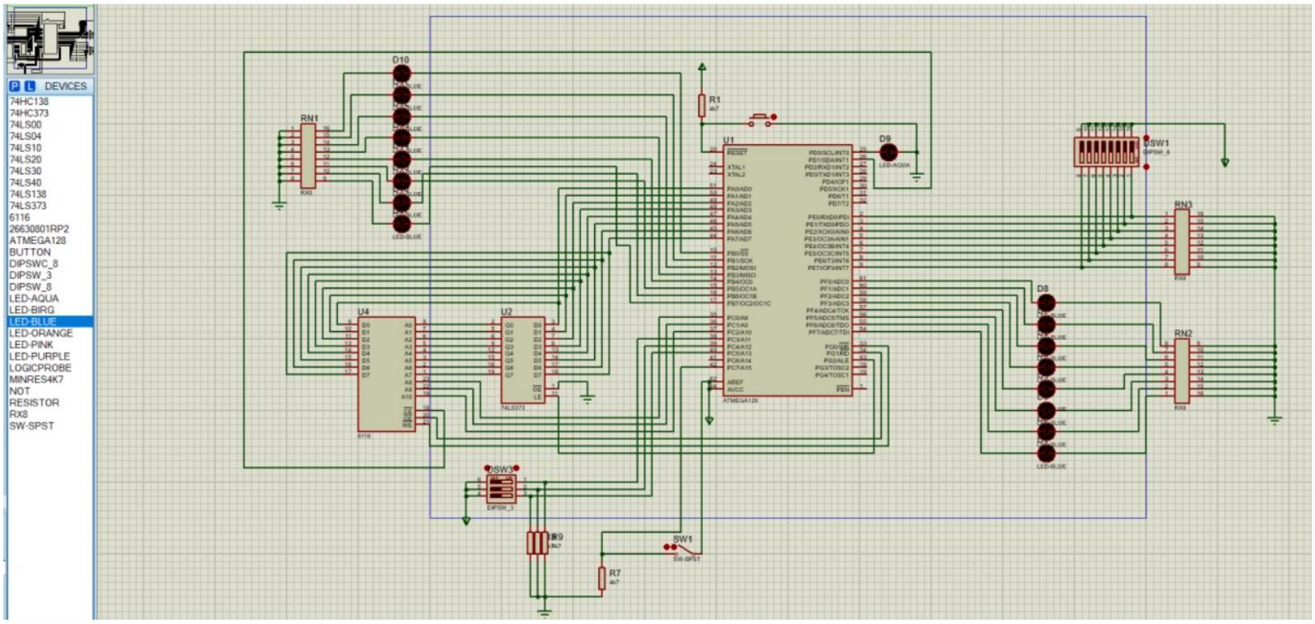
```
DUMP:
    LD R3, X+            ; load the value in X register into R3
    OUT PORTB, R3        ; output R3 to PORTB
    CALL DELAY           ; call DELAY subroutine
    Sbic PINC, 4         ; if memory dump isn't clear, then jump to MEM_DUMP. if clear, return back to the main program
    RJMP MEM_DUMP        ; repeat dumping process
    RET

LAST_ENTRY:
    Cpi ZH, HIGH(MEM_START)     ; if address is MEM_START, read MEM_START address
    BRNE CONTINUE               ; otherwise continue with normal case
    Cpi ZL, LOW(MEM_START)
    BRNE CONTINUE
    LD R3, Z                    ; load Z into R3
    OUT PORTB, R3               ; output R3 to PORTB
    RET
    CONTINUE:       ; decrement address, read the value there and then increment the address back
            Sbiw R31:R30, 1     ; decrement Z register address
            LD R3, Z            ; load Z content into R3
            OUT PORTB, R3       ; output R3 to PORTB
            Adiw R31:R30, 1     ; increment Z address back
            RET

DELAY:
    Ldi R29, 0xFF           ; load ONES to R29
    Ldi R28, 0xFF           ; load ONES to R28
    Ldi COUNTER, 0x04       ; load 4 for the COUNTER value
    L1: DEC COUNTER         ; decrement COUNTER
        BREQ DONE           ; if COUNTER is not 0, then continue. if 0, then branch to DONE and so return the main program
    L2: DEC R29             ; decrement R29 value
        Cpi R29, 0          ; check if R29 is 0. if yes, then go to L1. if not, then continue with L3 (execute the next lir
        BREQ L1
    L3: DEC R28             ; decrement R28 value
        Cpi R28, 0          ; check if the R28 reached 0
        BRNE L3             ; if not, then branch to L3 again to continue the same process until it's 0
        RJMP L2             ; if R28=0, then jump to L2

DONE: RET
```

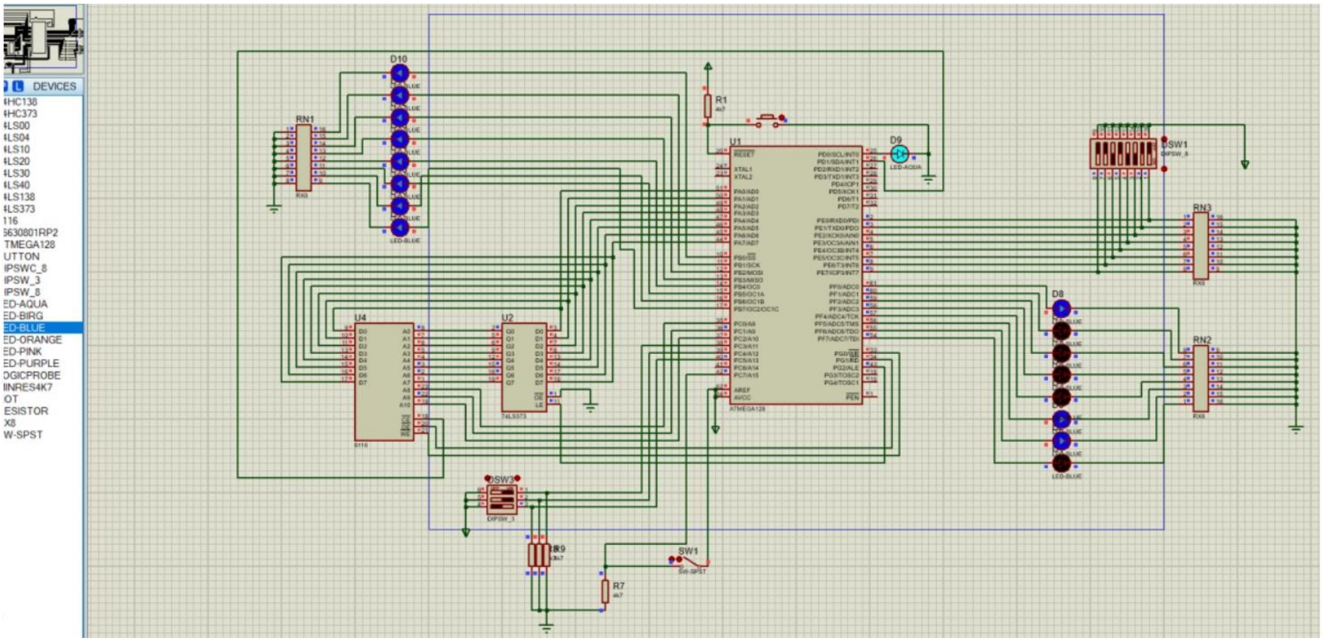
Name	Value
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x00
R17	0xFF
R18	0x00
R19	0x00
R20	0x00
R21	0x18
R22	0x00
R23	0x05
R24	0x00
R25	0x00
R26	0x01
R27	0x01

Name	Address	Value	Bits
PINF	0x20	0x00	□□□□□□□□
PINE	0x21	0x5F	□■□■■■■■
DDRE	0x22	0x00	□□□□□□□□
PORTE	0x23	0x00	□□□□□□□□
PIND	0x30	0xFE	■■■■■■■□
DDRD	0x31	0x03	□□□□□□■■
POR...	0x32	0x02	□□□□□□■□
PINC	0x33	0x18	□□□■■□□□
DDRC	0x34	0x18	□□□■■□□□
PORTC	0x35	0x18	□□□■■□□□
PINB	0x36	0x00	□□□□□□□□
DDRB	0x37	0xFF	■■■■■■■■
PORTB	0x38	0x00	□□□□□□□□
PINA	0x39	0x01	□□□□□□□■
DDRA	0x3A	0x00	□□□□□□□□
PORTA	0x3B	0x00	□□□□□□□□
DDRF	0x61	0xFF	■■■■■■■■
PORTF	0x62	0x00	□□□□□□□□

2.5 EXPERIMENTAL WORK

-> 0xD3 is tested as a packet_in value with mem_dump on condition; as a result, all readout LEDs are turned on.


DEVICES
74HC138
74HC373
74LS00
74LS04
74LS10
74LS20
74LS30
74LS40
74LS138
74LS373
6116
26630801RP2
ATMEGA128
BUTTON
DIPSWC_8
DIPSW_3
DIPSW_8
LED-AQUA
LED-BIRG
LED-BLUE
LED-ORANGE
LED-PINK
LED-PURPLE
LOGICPROBE
MINRES4K7
NOT
RESISTOR
RX8
SW-SPST
D10
RN1
R1
U1
D9
LED-AQUA
DSW1
RN3
D8
RN2
U4
U2
74LS373
ATMEGA128
6116
LED-BLUE
DSW3
R9
SW1
R7


DEVICES
4HC138
4HC373
4LS00
4LS04
4LS10
4LS20
4LS30
4LS40
4LS138
4LS373
116
5630801RP2
TMEGA128
UTTON
IPSWC_8
IPSW_3
IPSW_8
ED-AQUA
ED-BIRG
ED-BLUE
ED-ORANGE
ED-PINK
ED-PURPLE
OGICPROBE
IINRES4K7
OT
ESISTOR
X8
W-SPST
D10
RN1
R1
U1
D9
LED-AQUA
DSW1
RN3
D8
RN2
U4
U2
74LS373
ATMEGA128
6116
LED-BLUE
DSW3
R9
SW1
R7