Report

Laboratory 1

September 13, 2017

Author: Caroline Nilsson
Daniel Alm Grundström
Term: HT 2017
Course: 1DT301 - Computer
Technology I

Contents

1 Introduction

In the process of working with the laboratory assignments we started by doing research about the AVR Assembly language and the STK600 in order to better understand how to solve the different assignments. In each assignment we first created a pseudocode solution which we converted to flowchart diagrams, then it was rather simple to convert this into Assembly code. Common for all assignments is also that we have been using the simulations to confirm that the program is working and completing the correct tasks.

When we tested the Assembly programs on hardware we had trouble with the LEDs outputting inversed values. For example the program for assignment 1, which is supposed light `LED2`, lit all LEDs except `LED2`. If we understood this correctly, this was due to the pull-up resistor being activated on `PORTB` which made the LEDs light when the corresponding bit on `PORTB` was 0 (as opposed to 1). We fixed this by changing the bit string values we wrote to the LEDs in each assignment.

Another small issue we encountered was that we had assumed that `PORTD` would be the port connected to the switches on the STK600, as this is the configuration described in the STK600 User Guide. In the hardware we tested on though, the switches were connected to `PORTC`. To fix this we simply replaced all references of `PORTD`, `PIND` and `DDRD` to `PORTC`, `PINC` and `DDRC` in our Assembly code.

2 Assignment 1 - Light LED2

In the first assignment we were to write an Assembly program that lights up `LED2` (which is the third light counting from the right).

2.1 Pseudo code

Algorithm 1 Light LED2

---
    procedure PSEUDOCODE
        $PortB = output$
        $Led2\ bitstring \rightarrow PortB$

2.2 Flowchart
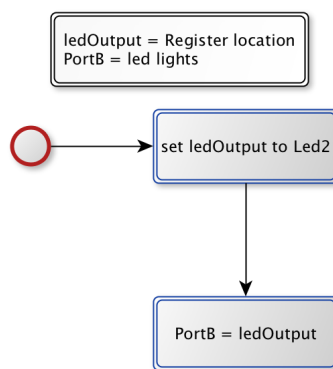
Figure 1: Basic flow in order to light LED2

2.3 Method

The pseudocode (see algorithm 1) and the flowchart (see figure 1) shows that we first set `PORTB` as an output port. To light up `LED2` we then only need to write a 0 to the bit on `PORTB` that corresponds to `LED2`.

    The minimal number of lines required to write this program we think are four. Two lines are required to set the LED port as output: one to write a value to a register and the other to write that value to the data direction register. The final two lines are for turning on the LED: one to write the LED state to a register and the other to write the LED state to the output port. One could try to write the program in three lines, by reusing the value written to the data direction register when writing to the output port. But in this case the LED will not turn on because of the pull-up resistor which will require that a zero is written to the bit corresponding to the LED that we want to light.

2.4 Assembly Program

```
1    ;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
2    ;   1DT301, Computer Technology I
3    ;   Date: 2017-09-07
4    ;   Author:
5    ;                           Caroline Nilsson          (cn222nd)
6    ;                           Daniel Alm Grundström      (dg222dw)
7    ;
8    ;   Lab number:             1
9    ;   Title:                  How to use the PORTs. Digital input/output.
10   ;                           Subroutine call.
11   ;
12   ;   Hardware:               STK600, CPU ATmega2560
13   ;
14   ;   Function:               Lights LED2 on PORTB
15   ;
16   ;   Input ports:            N/A
17   ;
18   ;   Output ports:           PIN2 on PORTB
19   ;
20   ;   Subroutines:            N/A
21   ;   Included files:         m2560def.inc
22   ;
23   ;   Other information:  LEDs are configured to light when PINs on PORTB are set
24   ;                       to 0. The default state, when no LED is lit must
25   ;                       therefore be set to 0b1111_1111.
26   ;
27   ;   Changes in program:
28   ;                           2017-09-01:
29   ;                           Implemented flowchart design.
30   ;
31   ;                           2017-09-02:
32   ;                           Added comments and .def for r16
33   ;
34   ;                           2017-09-07:
35   ;                           Adjusts code to handle pull-up resistors on PORTB.
36   ;                           Removes unnecessary loop that prevented program from
37   ;                           exiting after LED2 had been turned on.
38   ;
39   ;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
40   .include "m2560def.inc"
41   .def ledOutput = r16
42
43   ; Set PORTB to output
44   ldi ledOutput, 0xFF
45   out DDRB, ledOutput
46
47   ; Turn on LED2 on PORTB
48   ldi ledOutput, 0b1111_1011
49   out PORTB, ledOutput
```

3 Assignment 2 - Switch light corresponding LED

In the second assignment we were to write a program that waits for a switch to be pressed and then lights up the corresponding LED. For example if switch 3 is pressed LED 3 should light up. The way we interpreted the assignment was that the LED should stay on for as long as the switch is pressed down and turn off when the switch is released.

3.1 Pseudo code

Algorithm 2 Switches pressed lights corresponding LED
---
procedure PSEUDOCODE
    $PortB = output$
    $PortC = input$
    repeat
        $PortC\ value \rightarrow switchState$ $\triangleright switchState = register\ location$
        $switchState \rightarrow PortB$
    until ∞

3.2 Flowchart

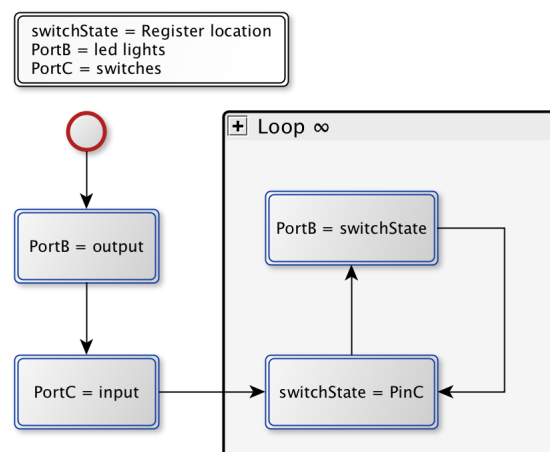Figure 2: Basic flow in order to read switches and light corresponding LED

3.3 Method

We figured the simplest way to solve this problem was to redirect the input from the switches to the LEDs repeateadly in a loop. We start by setting `PORTB` as output and `PORTC` as input. Then, in an infinite loop, we read the value of the pins on `PORTC` to a register and output the value of that register to `PORTB`.

3.4 Assembly Program

```asm
1    ;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
2    ;    1DT301, Computer Technology I
3    ;    Date: 2017−09−07
4    ;    Author:
5    ;                        Caroline Nilsson          (cn222nd)
6    ;                        Daniel Alm Grundström      (dg222dw)
7    ;
8    ;    Lab number:        1
9    ;    Title:             How to use the PORTs. Digital input/output.
10   ;                       Subroutine call.
11   ;
12   ;    Hardware:          STK600, CPU ATmega2560
13   ;
14   ;    Function:          Reads input from the switches SW0..SW7 and lights the
15   ;                       corresponding LED when a switch is pressed. (SW0 lights
16   ;                       LED0, SW1 lights LED1 and so on)
17   ;
18   ;    Input ports:       PORTC
19   ;
20   ;    Output ports:      PORTB
21   ;
22   ;    Subroutines:       N/A
23   ;    Included files:    m2560def.inc
24   ;
25   ;    Other information: N/A
26   ;
27   ;    Changes in program:
28   ;                       2017−09−01:
29   ;                       Implemented flowchart design.
30   ;
31   ;                       2017−09−02:
32   ;                       Adds header and comments.
33   ;
34   ;                       2017−09−07:
35   ;                       Adjusts code to handle pull up resistor on PORTB.
36   ;                       Changes switch port to PORTC.
37   ;
38   ;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
39   .include "m2560def.inc"
40   .def switchInput = r16
41   .def dataDir = r17
42
43   ; Set PORTB (LEDs) as output
44   ldi dataDir, 0xFF
45   out DDRB, dataDir
46
47   ; Set PORTC (switches) as input
48   ldi dataDir, 0x00
49   out DDRC, dataDir
50
51   loop:
52       in switchInput, PINC        ; Read input from switches
53       out PORTB, switchInput      ; Output switch input to LEDs
54       rjmp loop
```

4 Assignment 3 - Switch 5 lights LED0

In the third assignment we were to write an Assembly program that turns on `LED0` when the switch `SW5` is pressed. Nothing should happen when the other switches are pressed. We assumed that the way the program should work is that the LED would stay lit for as long as `SW5` was pressed down and turn off when the switch was released.

4.1 Pseudo code

Algorithm 3 Light LED0 when switch5 is pressed

---
procedure PSEUDOCODE
    $PortB = output$
    $PortC = input$
    repeat
        $reset\ ledState$ $\triangleright\ ledState = register\ location$
        if $Switch5\ is\ pressed$ **then**
            $ledState = LED0\ bit\ string$
        $ledState \rightarrow PortB$
    until ∞

4.2 Flowchart
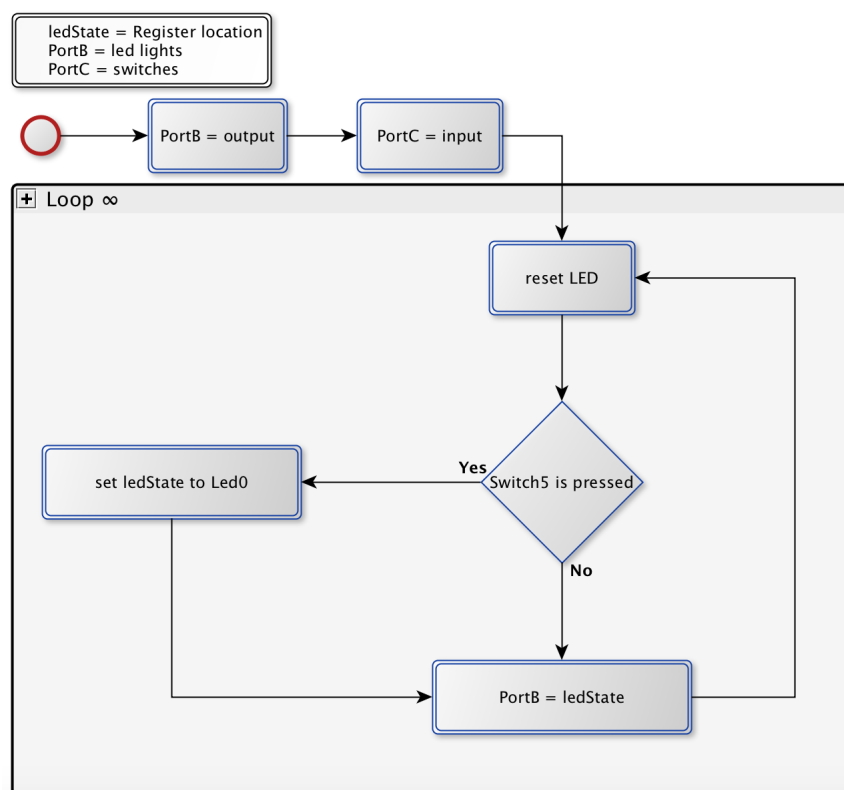
Figure 3: Basic flow in order to light LED 0 when switch 5 is pressed

4.3 Method

As with the previous assignment, we figured the first thing that needed to be done was to set the LED port, PORTB, as output and the switch port, PORTC, as input. In a loop, we then reset the value to write to the leds before checking if SW5 is pressed down. If the switch is pressed down we clear the least significant bit in the LED output value to indicate that we want LED0 to turn on before writing the value out to the LEDs.

4.4 Assembly Program

```
1    ;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
2    ;   1DT301, Computer Technology I
3    ;   Date: 2017-09-07
4    ;   Author:
5    ;                       Caroline Nilsson         (cn222nd)
6    ;                       Daniel Alm Grundström     (dg222dw)
7    ;
8    ;   Lab number:         1
9    ;   Title:              How to use the PORTs. Digital input/output.
10   ;                       Subroutine call.
11   ;
12   ;   Hardware:           STK600, CPU ATmega2560
13   ;
14   ;   Function:           Turns on LED0 when SW5 is held down.
15   ;
16   ;   Input ports:        PORTC
17   ;
18   ;   Output ports:       PORTB
19   ;
20   ;   Subroutines:        N/A
21   ;   Included files:     m2560def.inc
22   ;
23   ;   Other information:  N/A
24   ;
25   ;   Changes in program:
26   ;                       2017-09-01:
27   ;                       Implemented flowchart design.
28   ;
29   ;                       2017-09-04:
30   ;                       Minor refactoring. Adds header and comments.
31   ;
32   ;                       2017-09-07:
33   ;                       Adjusts code to handle pull up resistor on PORTB.
34   ;                       Changes switch port to PORTC.
35   ;
36   ;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
37   .include "m2560def.inc"
38   .def dataDir = r16
39   .def ledState = r17
40
41   ; Set PortB as output
42   ldi dataDir, 0xFF
43   out DDRB, dataDir
44
45   ; Set PortC as input
46   ldi dataDir, 0x00
47   out DDRC, dataDir
48
49   loop:
50       ser ledState                    ; Set bits in LED state so LEDs are turned
51                                       ; off when button is released
52
53       sbis PINC, PINC5                ; If SW5 is pressed down (PINC5 bit is zero)
54           ldi ledState, 0xFE          ;   then set LED0 state to turned on
55
56       out PORTB, ledState             ; write state to LEDs
57       rjmp loop
```

5 Assignment 4 - Using the AVR simulator

For this assignment, we were to take the program we wrote for the previous assignment, that lights LED0 when SW5 is pressed down, and run it in the AVR simulator that is included in Atmel Studio.

5.1 Method

First, we needed to setup the debugger to run the program on the simulator which is done by selecting "Simulator" in *Project → Properties → Tool*. To start stepping through the program we then clicked *Debug → Start debugging and break*. To be able to see what happens in the simulator as the program runs, we opened the *Processor Status* and *I/O* windows by clicking on their respective icons. Finally, because values from the pins on the switches have a default value of 1 we needed to set all PINC bits, which we did by selecting *I/O Port (PORTC)* in the I/O window and filling in the bits to the right of *PINC* by clicking on them.

We then started stepping through the program. The first few lines sets PORTB as output (line 42-43) and PORTC to input (line 46-47) and this can be seen in the simulator by keeping an eye on *PORTB → DDRB*, which bits are set to 1, and *PORTC → DDRC*, which are set to 0, as the debugger executes the instructions. When the debugger executes line 50, which sets all bits in register 17, we can see this in the Processor Status window in that *r17* gets set to 0xFF. We tested that the instruction on line 53, which checks if SW5 is pressed down, works by manually clearing bit 5 on PINC through the I/O window. The instruction that gets executed if this is true, can be seen in the simulator in that register 17 changes value from 0xFF to 0xFE. The result of line 56, which writes the value of register 17 to PORTB, can be seen by clicking on PORTB in the I/O window where the bits next to PORTB and PINB are updated accordingly. We could also see that the *Program Counter* value in the Processor Status window gets updated when the final line rcall loop gets executed.

6 Assignment 5 - Waterfall

6.1 Assignment description

The fifth assignment was to write an Assembly program that outputs a ring counter to the LEDs. Between each value in the counter, there should be a delay of approximately 0.5 seconds. Since the delay should be a subroutine in the program, the stack pointer SP needs to be initialized as instructed in the description of the assignment.

6.2 Pseudo code

Algorithm 4 Display Ring counter on LEDs
---
procedure PSEUDOCODE
    Initialize stack pointer
    $PortB = output$
    Initialize ledState $\triangleright ledState = register\ location$
    repeat
        $ledState \rightarrow PortB$
        Delay
        rotate ledState to left
    until ∞

6.3 Flowchart
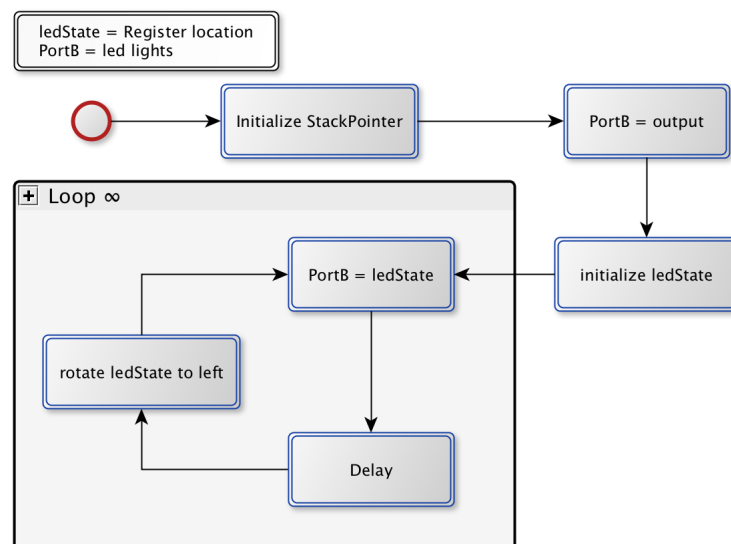
Figure 4: Basic flow in order to display a Ring counter on the LEDs

6.4 Method

The first thing we needed to do, as described above, was to initialize the stack pointer. This is done by setting SP to the end of SRAM (RAMEND). Since SP is a 16-bit register we needed to set both SPL and SPH. SPL is set to the least significant 8 bits of RAMEND and SPH is set to the most significant 8 bits of RAMEND. As always, we also set PORTB as output so we can write values to the LEDs.

The main part of the program consists of a loop where the current LED state is first written to the LEDs. We then delay execution of the program for ~0.5 seconds and finally rotate the bits in the LED state to the left using the rol instruction.

The delay functionality is as described in the assignment description implemented in a subroutine which we have calculated using the *AVR Delay Loop Calculator*.[1] We calculated a delay of 500 ms for 1.0 MHz, which is the default clock speed of the AVR ATmega2560.[2] We modified the subroutine slightly to push the registers that are used in it to the stack at the start of the subroutine and pop them before returning. This is done so we do not accidentally overwrite any values in the registers that the subroutine is using. These additional instructions do of course make the delay slightly longer, particularly since the CPU will need to write to and read from SRAM, but we did not think that mattered that much in this case.

---

[1]   http://www.bretmulvey.com/avrdelay.html

[2]   *Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V DATASHEET*, Atmel Corporation, San Jose, CA, 2014, pp. 40

6.5 Assembly Program

```
1    ;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
2    ;    1DT301, Computer Technology I
3    ;    Date: 2017-09-07
4    ;    Author:
5    ;                          Caroline Nilsson            (cn222nd)
6    ;                          Daniel Alm Grundström       (dg222dw)
7    ;
8    ;    Lab number:           1
9    ;    Title:                How to use the PORTs. Digital input/output.
10   ;                          Subroutine call.
11   ;
12   ;    Hardware:             STK600, CPU ATmega2560
13   ;
14   ;    Function:             Repeatedly lights LEDs sequentially right to left.
15   ;
16   ;                          I.e:
17   ;                          0000 0001 -> 0000 0010 -> 0000 0100 -> ... ->
18   ;                          1000 0000 -> 0000 0001 -> 0000 0010 -> ...
19   ;
20   ;    Input ports:          N/A
21   ;
22   ;    Output ports:         PORTB
23   ;
24   ;    Subroutines:          delay - delays execution
25   ;    Included files:       m2560def.inc
26   ;
27   ;    Other information:    Since a subroutine is used, the stack pointer must
28   ;                          be initialized so the processor knows where in the
29   ;                          code to jump when the subroutine returns.
30   ;
31   ;    Changes in program:
32   ;                          2017-09-01:
33   ;                          Implements flowchart design
34   ;
35   ;                          2017-09-04:
36   ;                          Adds header, comments and some minor refactoring
37   ;
38   ;                          2017-09-07:
39   ;                          Adjusts code to handle pull up resistor on PORTB.
40   ;
41   ;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
42   .include "m2560def.inc"
43   .def dataDir = r16
44   .def ledState = r17
45   .equ INITIAL_LED_STATE = 0xFF
46
47   ; Initialize SP, Stack Pointer
48   ldi r20, HIGH(RAMEND)                  ; R20 = high part of RAMEND address
49   out SPH, R20                           ; SPH = high part of RAMEND address
50   ldi R20, low(RAMEND)                   ; R20 = low part of RAMEND address
51   out SPL, R20                           ; SPL = low part of RAMEND address
52
53   ; Set PORTB to output
54   ldi dataDir, 0xFF
55   out DDRB, dataDir
56
57   ldi ledState, INITIAL_LED_STATE        ; Set initial LED state
58
59   loop:
60       out PORTB, ledState                ; Write state to LEDs
61       rcall delay                        ; Delay to make changes visible
62       rol ledState                       ; Rotate LED state to the left
63       rjmp loop
64
65   ; Generated by delay loop calculator
66   ; at http://www.bretmulvey.com/avrdelay.html
67   ;
68   ; Delay 500 000 cycles
69   ; 500ms at 1 MHz
70   delay:
71       push r18
72       push r19
73       push r20
74
75       ldi r18, 3
76       ldi r19, 138
77       ldi r20, 86
78   L1: dec r20
79       brne L1
80       dec r19
81       brne L1
82       dec r18
83       brne L1
84       rjmp PC+1
85
86       pop r20
87       pop r19
88       pop r18
89       ret
```

12

7   Assignment 6 - Johnson counter

The final assignment was to write a program that displays a *Johnson counter* to the LEDs in an infinite loop. A Johnson counter is a counter that sets all bits in a bit string one-by-one and, when all bits are set, clears the bits one-by-one. Of course, this program will also need a delay after each value in the Johnson counter has been written to the LEDs. Otherwise the program would run so fast that all LEDs would most likely appear to be lit all the time.

7.1   Pseudo code

Algorithm 5 Johnson counter simulation using LEDs

$\quad$ **procedure** PSEUDOCODE
$\qquad PortB = output$ $\hfill \triangleright complement = register\ location$
$\qquad Initialize\ currentValue$ $\hfill \triangleright currentValue = register\ location$
$\qquad$ **repeat** $\hfill \triangleright Loop\_1\ (count\ up)$
$\qquad\quad$ **if** $LED7\ is\ lit$ **then**
$\qquad\qquad Continue\ at\ Loop\_2$
$\qquad\quad$ **else**
$\qquad\qquad currentValue = currentValue \times 2$
$\qquad\qquad Increase\ currentValue\ by\ 1$
$\qquad\qquad complement = complement\ of\ currentValue$
$\qquad\quad complement \rightarrow PortB$
$\qquad\quad Delay$
$\qquad$ **until** ∞
$\qquad$ **repeat** $\hfill \triangleright Loop\_2\ (count\ down)$
$\qquad\quad$ **if** $LED0\ is\ lit$ **then**
$\qquad\qquad Continue\ at\ Loop\_1$
$\qquad\quad$ **else**
$\qquad\qquad currentValue = Shift\ right$
$\qquad\qquad complement = complement\ of\ currentValue$
$\qquad\quad complement \rightarrow PortB$
$\qquad\quad Delay$
$\qquad$ **until** ∞

7.2 Flowchart
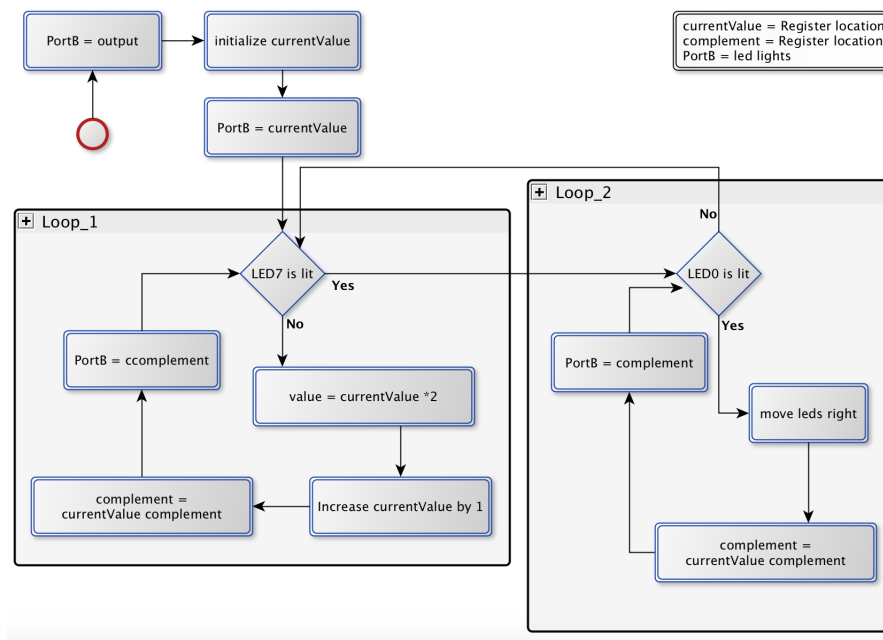
Figure 5: Basic flow in order to display a Johnson counter on LEDs

7.3 Method

We started by trying to figure out the mathematical formula for finding Johnson values. In decimal, the values for an 8-bit Johnson counter are: 0, 1, 3, 7, 15, 31, 63, 127 and 255. We realized that we could get the next value by multiplying by 2 and adding 1, which gives the recurrence relation

$$J_n = J_{n-1} \cdot 2 + 1, \qquad n \in \mathbb{N}_0$$

With this we could get both the next and previous Johnson value. In the case of getting the previous value, we considered that we are dealing with integers, which because of truncating means we only needed to divide the current value by 2 to count down the counter.

As with assignment 5, we start by setting the stack pointer since we are going to use a subroutine for the delay function. We also set `PORTB` as output. To count the Johnson counter up and down, we created two loops: `count_up` and `count_down`. In `count_up` we multiply the current counter value by 2 by shifting it to the left and then increment it by 1 before outputting the value to the LEDs. We repeat this until all the LEDs are lit, upon which executions jumps to `count_down`. In this loop, we divide the current counter value by 2 by shifting it to the right and then output the value to the LEDs. This is in turn repeated until all the LEDs are turned off, upon which the execution jumps back to `count_up` and the process starts again.

14

7.4 Assembly Program

```
1   ;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
2   ;   1DT301, Computer Technology I
3   ;   Date: 2017-09-07
4   ;   Author:
5   ;                       Caroline Nilsson         (cn222nd)
6   ;                       Daniel Alm Grundström    (dg222dw)
7   ;
8   ;   Lab number:         1
9   ;   Title:              How to use the PORTs. Digital input/output.
10  ;                       Subroutine call.
11  ;
12  ;   Hardware:           STK600, CPU ATmega2560
13  ;
14  ;   Function:           Lights LEDs as a Johnson counter in an infinite loop.
15  ;
16  ;                       I.e:
17  ;                       0000 0001 -> 0000 0011 -> 0000 0111 -> ...
18  ;                       1111 1111 -> 0111 1111 -> 0011 1111 -> ...
19  ;
20  ;   Input ports:        N/A
21  ;
22  ;   Output ports:       PORTB
23  ;
24  ;   Subroutines:        delay - delay execution
25  ;
26  ;   Included files:     m2560def.inc
27  ;
28  ;   Other information:  N/A
29  ;
30  ;   Changes in program:
31  ;                       2017-09-02:
32  ;                       Implements flowchart design
33  ;
34  ;                       2017-09-04:
35  ;                       Adds header and comments
36  ;
37  ;                       2017-09-07:
38  ;                       Adjusts code to handle pull up resistor on PORTB.
39  ;                       Changes code to use shift left instead of multiplying
40  ;
41  ;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
42  .include "m2560def.inc"
43  .def dataDir = r16
44  .def currentValue = r17           ; Current value of Johnson counter
45  .def complement = r18
46
47  ; Initialize SP, Stack Pointer
48  ldi r20, HIGH(RAMEND)             ; R20 = high part of RAMEND address
49  out SPH,R20                       ; SPH = high part of RAMEND address
50  ldi R20, low(RAMEND)              ; R20 = low part of RAMEND address
51  out SPL,R20                       ; SPL = low part of RAMEND address
52
53  ; Set PORTB as output
54  ldi dataDir, 0xFF
55  out DDRB, dataDir
56
57  ; Set and output initial value
58  ldi currentValue, 0x00
59  rcall led_out
60
61  count_up:
62      sbis PORTB, PINB7             ; If LED7 is lit (i.e. all LEDs lit)
63          rjmp count_down          ;     then start counting down
64
65      ; Get next johnson value by multiplying by 2 and adding 1
66      lsl currentValue
67      inc currentValue
68
69      rcall led_out                ; Output complement of current value
70      rcall delay_500ms            ; Delay to make changes visible
71      rjmp count_up                ; Continue counting up
72
73  count_down:
74      sbic PORTB, PINB0            ; If LED0 is unlit (i.e. all LEDs unlit)
75          rjmp count_up            ;     then start counting up
76
77      lsr currentValue             ; Shift to right to get previous
78                                   ; johnson value
79
80      rcall led_out                ; Ouput complement of current value
81      rcall delay_500ms            ; Delay to make changes visible
82      rjmp count_down              ; Continue counting down
83
84  ; Writes the complement of 'currentValue' to PORTB
85  led_out:
86      mov complement, currentValue
87      com complement
88      out PORTB, complement
89      ret
90
91  ; Generated by delay loop calculator
92  ; at http://www.bretmulvey.com/avrdelay.html
93  ;
94  ; Delay 500 000 cycles
95  ; 500ms at 1 MHz
96  delay:
97      push r18
98      push r19
99      push r20
```

15

```
100
101          ldi   r18 ,  3
102          ldi   r19 ,  138
103          ldi   r20 ,  86
104   L1 :  dec   r20
105          brne  L1
106          dec   r19
107          brne  L1
108          dec   r18
109          brne  L1
110          rjmp  PC+1
111
112          pop   r20
113          pop   r19
114          pop   r18
115          ret
```