Report

Laboratory 1

September 10, 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.

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

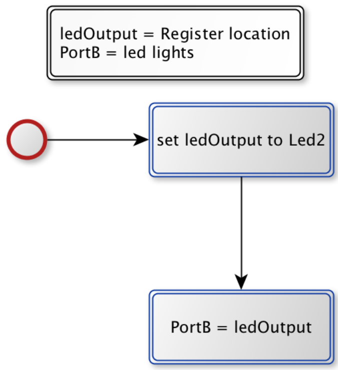

Figure 1: Flowchart

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 value to the bit on `PORTB` that corresponds to `LED2`.

We started with the assumption that all bits in `PORTB` would be zero when the LEDs where turned off and as such wrote a 1 to the third least significant bit to light up `LED2`. When we tested the program on the hardware however, all LEDs except `LED2` was turned on. If we understood this correctly, this was due to the pull-up resistor being activated on `PORTB` which made the LEDs light when their bit was 0 (as opposed to 1) on `PORTB`. We fixed this by simply inverting the value we wrote to `PORTB` ($1111\ 1011_2$ instead of $0000\ 0100_2$).

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

```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;	1DT301, Computer Technology I
;	Date: 2017-09-07
;	Author:
;				Caroline Nilsson		(cn222nd)
;				Daniel Alm Grundström	(dg222dw)
;
;	Lab number:		1
;	Title:			How to use the PORTs. Digital input /output.
;				Subroutine call.
;
;	Hardware:		STK600, CPU ATmega2560
;
;	Function:		Lights LED2 on PORTB
;
;	Input ports:		N/A
;
;	Output ports:		PIN2 on PORTB
;
;	Subroutines:		N/A
;	Included files:		m2560def.inc
;
;	Other information:	LEDs are configured to light when PINs on PORTB are set
;				to 0. The default state, when no LED is lit must
;				therefore be set to 0b1111_1111.
;
;	Changes in program:
;				2017-09-01:
;				Implemented flowchart design.
;
;				2017-09-02:
;				Added comments and .def for r16
;
;				2017-09-07:
;				Adjusts code to handle pull-up resistors on PORTB.
;				Removes unnecessary loop that prevented program from
;				exiting after LED2 had been turned on.
;
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
.include "m2560def.inc"
.def ledOutput = r16

; Set PORTB to output
ldi ledOutput, 0xFF
out DDRB, ledOutput

; Turn on LED2 on PORTB
ldi ledOutput, 0b1111_1011
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
    $PortC\ value \rightarrow switchState$ ▷ $switchState = register\ location$
    $switchState \rightarrow PortB$
  until ∞

3.2 Flowchart

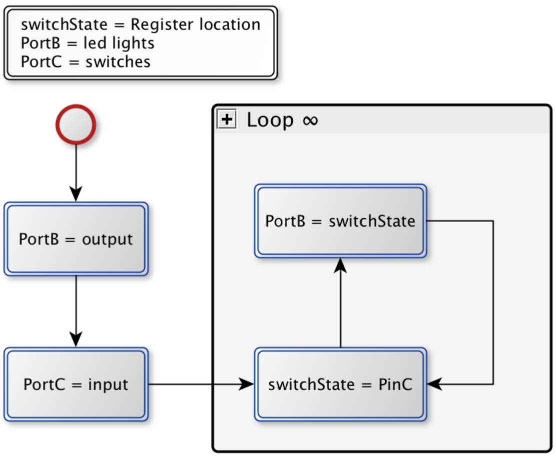

Figure 2: Basic flow in order to read switches and light corresponding LED

3.3 Method

We figured the simplest way to solve this problem was to simply redirect the input from the switches to the LEDs repeateadly in a loop. Initially we thought that we hade to take the complement of the input since we assumed that the switches used a pull-up resistor and the LEDs did not. For example if `SW5` was pressed, the input would be $1101\ 1111_2$ but the output would need to be $0010\ 0000_2$ to light up `LED5`. When the program was tested on hardware we noticed that this was not the case and as such there was no need to take the complement of the input.

3.4 Assembly Program

```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;	1DT301, Computer Technology I
;	Date: 2017-09-07
;	Author:
;			Caroline Nilsson		(cn222nd)
;			Daniel Alm Grundström	(dg222dw)
;
;	Lab number:		1
;	Title:			How to use the PORTs. Digital input /output.
;			Subroutine call.
;
;	Hardware:		STK600, CPU ATmega2560
;
;	Function:		Reads input from the switches SW0..SW7 and lights the
;			corresponding LED when a switch is pressed. (SW0 lights
;			LED0, SW1 lights LED1 and so on)
;
;	Input ports:		PORTC
;
;	Output ports:		PORTB
;
;	Subroutines:		N/A
;	Included files:	m2560def.inc
;
;	Other information:	N/A
;
;	Changes in program:
;			2017-09-01:
;			Implemented flowchart design.
;
;			2017-09-02:
;			Adds header and comments.
;
;			2017-09-07:
;			Adjusts code to handle pull up resistor on PORTB.
;			Changes switch port to PORTC.
;
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
.include "m2560def.inc"
.def switchInput = r16
.def dataDir = r17

; Set PORTB (LEDs) as output
ldi dataDir, 0xFF
out DDRB, dataDir

; Set PORTC (switches) as input
ldi dataDir, 0x00
out DDRC, dataDir

loop:
	in switchInput, PINC		; Read input from switches
	out PORTB, switchInput	; Output switch input to LEDs
	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
        $reset\ ledState$ ▷ $ledState = register\ location$
        if $Switch5\ is\ pressed$ **then**
            $ledState = LED0\ bit\ string$
        $ledState \rightarrow PortB$
    until ∞

4.2 Flowchart

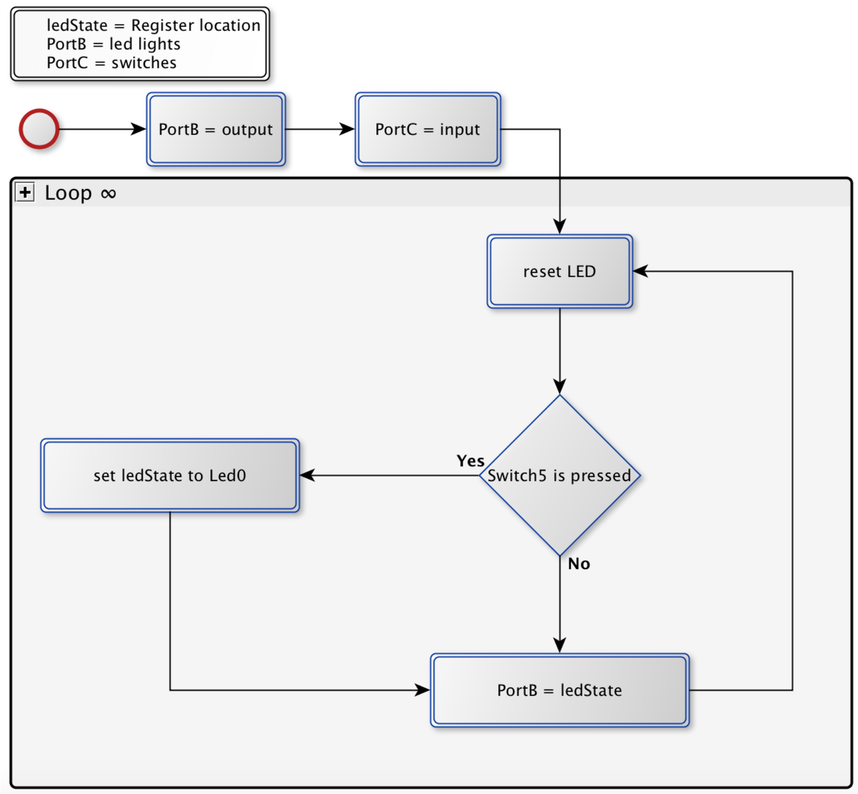

Figure 3: Flowchart

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.

When testing on hardware we needed to adjust the code for the pull-up resistors on the LEDs to get it to work.

4.4 Assembly Program

```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;    1DT301, Computer Technology I
;    Date: 2017-09-07
;    Author:
;                         Caroline Nilsson            (cn222nd)
;                         Daniel Alm Grundström       (dg222dw)
;
;    Lab number:          1
;    Title:               How to use the PORTs. Digital input /output.
;                         Subroutine call.
;
;    Hardware:            STK600, CPU ATmega2560
;
;    Function:            Turns on LED0 when SW5 is held down.
;
;    Input ports:         PORTC
;
;    Output ports:        PORTB
;
;    Subroutines:         N/A
;    Included files:      m2560def.inc
;
;    Other information:   N/A
;
;    Changes in program:
;                         2017-09-01:
;                         Implemented flowchart design.
;
;                         2017-09-04:
;                         Minor refactoring. Adds header and comments.
;
;                         2017-09-07:
;                         Adjusts code to handle pull up resistor on PORTB.
;                         Changes switch port to PORTC.
;
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
.include "m2560def.inc"
.def dataDir = r16
.def ledState = r17

; Set PortB as output
ldi dataDir, 0xFF
out DDRB, dataDir

; Set PortC as input
ldi dataDir, 0x00
out DDRC, dataDir

loop:
    ser ledState                    ; Set bits in LED state so LEDs are turned
                                    ; off when button is released

    sbis PINC, PINC5                ; If SW5 is pressed down (PINC5 bit is zero)
        ldi ledState, 0xFE          ;   then set LED0 state to turned on

    out PORTB, ledState             ; write state to LEDs
    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 Waterfall simulation using LEDs

procedure PSEUDOCODE
    $Initialize\ stack\ pointer$
    $PortB = output$
    $Initialize\ ledState$ ▷ $ledState = register\ location$
    repeat
        $ledState \rightarrow PortB$
        $Delay$
        $rotate\ ledState\ to\ left$
    until ∞

6.3 Flowchart

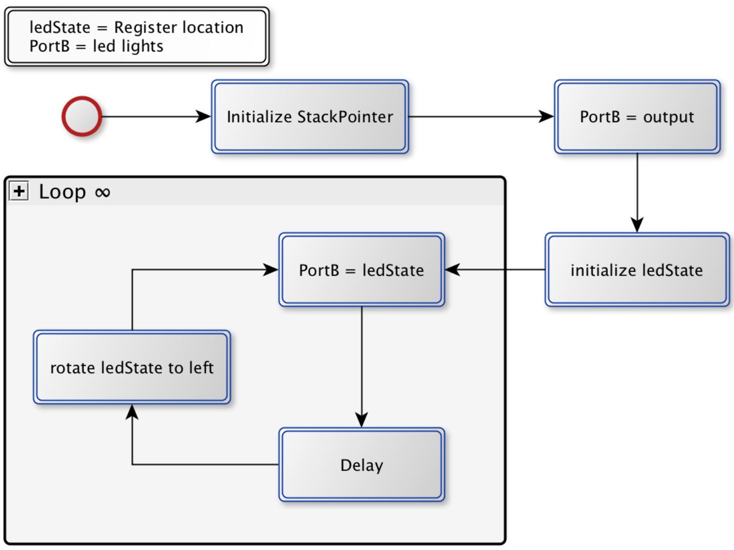

Figure 4: Flowchart

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 have 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.

As with the other assignments, when we tested the code on hardware we had to adjust it to handle the pull-up resistor on `PORTB`.

[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

```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;    1DT301, Computer Technology I
;    Date: 2017-09-07
;    Author:
;                          Caroline Nilsson           (cn222nd)
;                          Daniel Alm Grundström      (dg222dw)
;
;    Lab number:           1
;    Title:                How to use the PORTs. Digital input /output.
;                          Subroutine call.
;
;    Hardware:             STK600, CPU ATmega2560
;
;    Function:             Repeatedly lights LEDs sequentially right to left.
;
;                          I.e:
;                          0000 0001 -> 0000 0010 -> 0000 0100 -> ... ->
;                          1000 0000 -> 0000 0001 -> 0000 0010 -> ...
;
;    Input ports:          N/A
;
;    Output ports:         PORTB
;
;    Subroutines:          delay - delays execution
;    Included files:       m2560def.inc
;
;    Other information:    Since a subroutine is used, the stack pointer must
;                          be initialized so the processor knows where in the
;                          code to jump when the subroutine returns.
;
;    Changes in program:
;                          2017-09-01:
;                          Implements flowchart design
;
;                          2017-09-04:
;                          Adds header, comments and some minor refactoring
;
;                          2017-09-07:
;                          Adjusts code to handle pull up resistor on PORTB.
;
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
.include "m2560def.inc"
.def dataDir = r16
.def ledState = r17
.equ INITIAL_LED_STATE = 0xFF

; Initialize SP, Stack Pointer
ldi r20, HIGH(RAMEND)                   ; R20 = high part of RAMEND address
out SPH,R20                             ; SPH = high part of RAMEND address
ldi R20, low(RAMEND)                    ; R20 = low part of RAMEND address
out SPL,R20                             ; SPL = low part of RAMEND address

; Set PORTB to output
ldi dataDir, 0xFF
out DDRB, dataDir

ldi ledState, INITIAL_LED_STATE         ; Set initial LED state

loop:
    out PORTB, ledState                 ; Write state to LEDs
    rcall delay                         ; Delay to make changes visible
    rol ledState                        ; Rotate LED state to the left
    rjmp loop

; Generated by delay loop calculator
; at http://www.bretmulvey.com/avrdelay.html
;
; Delay 500 000 cycles
; 500ms at 1 MHz
delay:
    push r18
    push r19
    push r20

    ldi  r18, 3
    ldi  r19, 138
    ldi  r20, 86
L1: dec  r20
    brne L1
    dec  r19
    brne L1
    dec  r18
    brne L1
    rjmp PC+1

    pop r20
    pop r19
    pop r18
    ret
```

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

```
procedure PSEUDOCODE
    PortB = output                          ▷ complement = register location
    Initialize currentValue                 ▷ currentValue = register location
    repeat                                  ▷ Loop_1 (count up)
        if LED7 is lit then
            Continue at Loop_2
        else
            currentValue = currentValue × 2
            Increase currentValue by 1
            complement = complement of currentValue
        complement → PortB
        Delay
    until ∞
    repeat                                  ▷ Loop_2 (count down)
        if LED0 is lit then
            Continue at Loop_1
        else
            currentValue = Shift right
            complement = complement of currentValue
        complement → PortB
        Delay
    until ∞
```

7.2 Flowchart

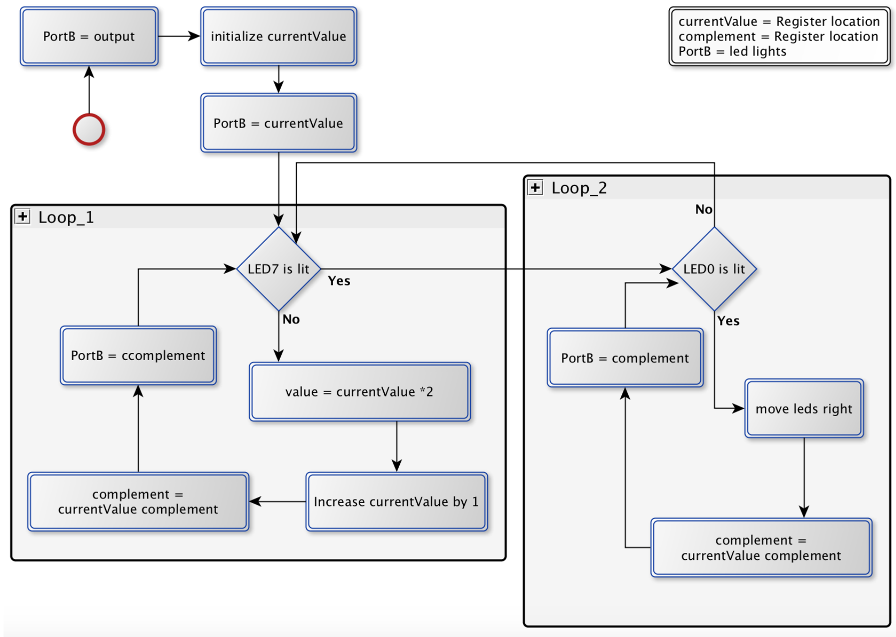

Figure 5: Flowchart

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.

When testing the program on hardware, we had some problems with the pull-up resistor causing the LEDs to output inverse states. We figured the easiest way to solve this was to create a subroutine, `led_out`, which gets the complement of the current Johnson value and outputs that to the LEDs.

7.4 Assembly Program

```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;    1DT301, Computer Technology I
;    Date: 2017-09-07
;    Author:
;                       Caroline Nilsson          (cn222nd)
;                       Daniel Alm Grundström     (dg222dw)
;
;    Lab number:        1
;    Title:             How to use the PORTs. Digital input/output.
;                       Subroutine call.
;
;    Hardware:          STK600, CPU ATmega2560
;
;    Function:          Lights LEDs as a Johnson counter in an infinite loop.
;
;                       I.e:
;                       0000 0001 -> 0000 0011 -> 0000 0111 -> ...
;                       1111 1111 -> 0111 1111 -> 0011 1111 -> ...
;
;    Input ports:       N/A
;
;    Output ports:      PORTB
;
;    Subroutines:       delay - delay execution
;
;    Included files:    m2560def.inc
;
;    Other information: N/A
;
;    Changes in program:
;                       2017-09-02:
;                       Implements flowchart design
;
;                       2017-09-04:
;                       Adds header and comments
;
;                       2017-09-07:
;                       Adjusts code to handle pull up resistor on PORTB.
;                       Changes code to use shift left instead of multiplying
;
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
.include "m2560def.inc"
.def dataDir = r16
.def currentValue = r17                 ; Current value of Johnson counter
.def complement = r18

; Initialize SP, Stack Pointer
ldi r20, HIGH(RAMEND)                   ; R20 = high part of RAMEND address
out SPH,R20                             ; SPH = high part of RAMEND address
ldi R20, low(RAMEND)                    ; R20 = low part of RAMEND address
out SPL,R20                             ; SPL = low part of RAMEND address

; Set PORTB as output
ldi dataDir, 0xFF
out DDRB, dataDir

; Set and output initial value
ldi currentValue, 0x00
rcall led_out

count_up:
    sbis PORTB, PINB7                   ; If LED7 is lit (i.e. all LEDs lit)
        rjmp count_down                 ;     then start counting down

    ; Get next johnson value by multiplying by 2 and adding 1
    lsl currentValue
    inc currentValue

    rcall led_out                       ; Output complement of current value
    rcall delay_500ms                   ; Delay to make changes visible
    rjmp count_up                       ; Continue counting up

count_down:
    sbic PORTB, PINB0                   ; If LED0 is unlit (i.e. all LEDs unlit)
        rjmp count_up                   ;     then start counting up

    lsr currentValue                    ; Shift to right to get previous
                                        ; johnson value

    rcall led_out                       ; Ouput complement of current value
    rcall delay_500ms                   ; Delay to make changes visible
    rjmp count_down                     ; Continue counting down

; Writes the complement of 'currentValue' to PORTB
led_out:
    mov complement, currentValue
    com complement
    out PORTB, complement
    ret

; Generated by delay loop calculator
; at http://www.bretmulvey.com/avrdelay.html
;
; Delay 500 000 cycles
; 500ms at 1 MHz
delay:
    push r18
    push r19
    push r20
```

```
        ldi   r18, 3
        ldi   r19, 138
        ldi   r20, 86
L1:     dec   r20
        brne  L1
        dec   r19
        brne  L1
        dec   r18
        brne  L1
        rjmp  PC+1

        pop r20
        pop r19
        pop r18
        ret
```