Name: Narasimhan Srikanth Roll Number: EE20B087

EE2016 – Microprocessor Lab Report

Experiment 2 – Interrupts in Atmel AVR Atmega

Target of the experiment:

The aim of the experiment is to use assembly language and C language to implement interrupts in Atmel AVR Atmega 8. In this experiment, a LED has to be blinked for 10 times (1 second switch on, then switch off). We use interrupts to make this work in assembly and C language.

Tasks:

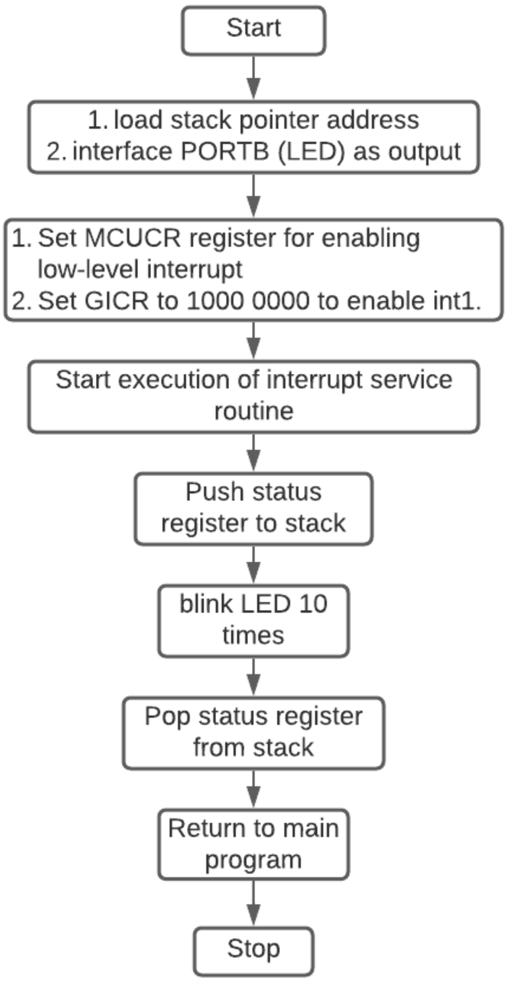
1. Fill in the blanks in the assembly code.

2. Use int0 to redo the same in the demo program (duly filled in). Once the switch is pressed the LED should blink 10 times.

3. Rewrite the program in 'C' (int1). Rewrite the C program for int0.

4. Demonstrate both the cases (of assembly and C).

Solution and code: for int1 implementation



When the ISR has loops in it, we have to make sure that the status register is pushed and popped before and after the ISR is executed (inside the ISR block of code). Otherwise, the contents in the status register could be affected by the loops.

The lowest memory address in the program memory space is by default reset and interrupt vector space. So, in Atmel AVR Atmega 8, 0x01 corresponds to int0 and 0x02 to int1.

In the program, I have used loops to blink LED ON for 0.5 seconds and OFF for 0.5 seconds.

Since an instruction cycle in Atmega 8 is 1 microsecond, we require 500000 microseconds of delay to keep LED on for 0.5 seconds. Since instructions like DEC and BRNE themselves take 3 cycles, I have chosen the values of registers such that the 3 nested loops together run for approximately 500000 machine cycles.

the assembly code is as follows,

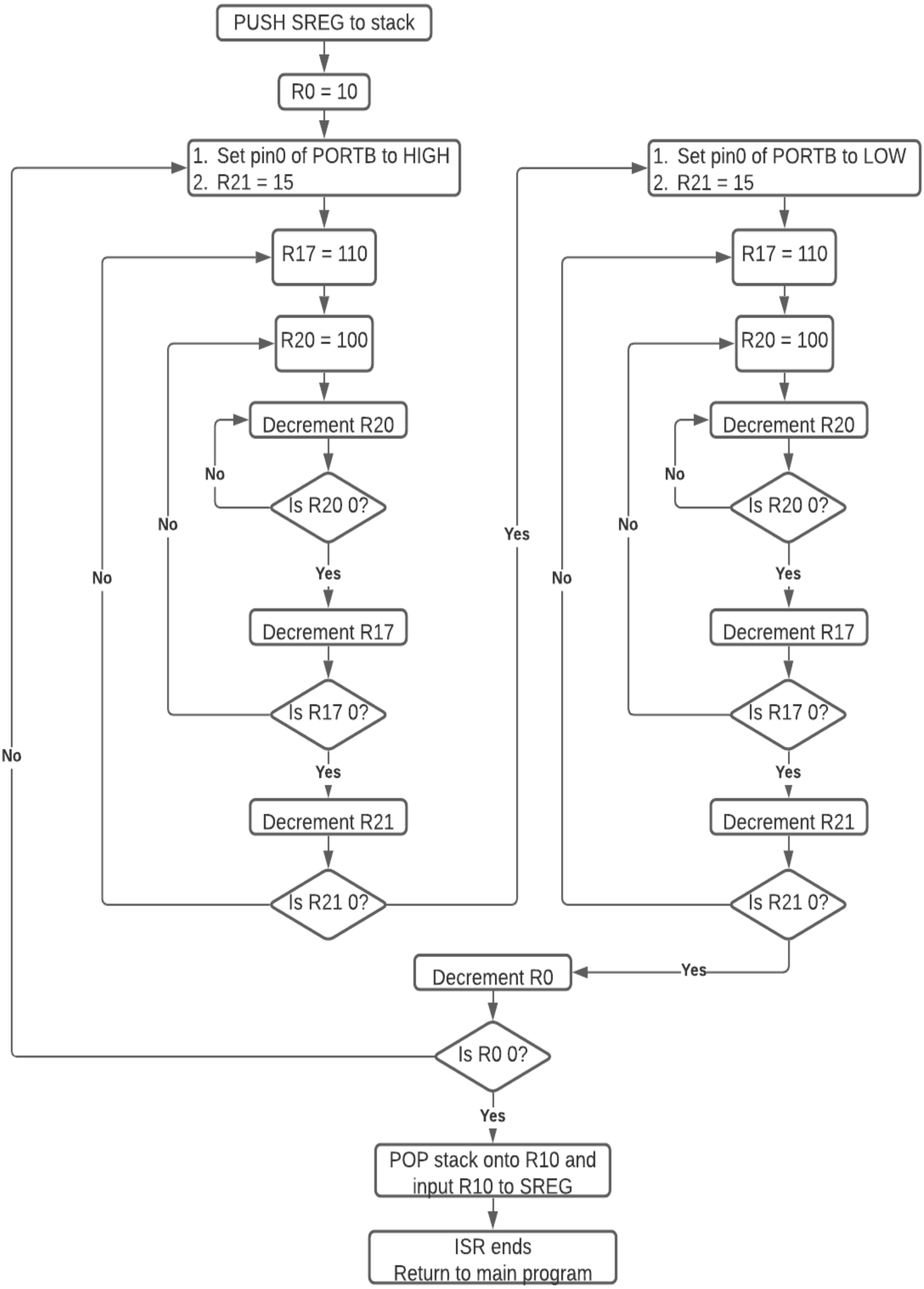
```
rjmp reset

.org 0x0002; interrupt vector for int1 in Atmega 8
rjmp int1_ISR

.org 0x0100

reset:
      LDI R16,0x70; Loading stack pointer address
         OUT SPL,R16
         LDI R16,0x00
         OUT SPH,R16
         LDI R16,0x00
         OUT DDRD,R16
         OUT MCUCR,R16; Set MCUCR register to enable low level interrupt
         LDI R16,0B10000000
         OUT GICR,R16; Set GICR register to 1000 0000 to enable int1
         LDI R16,0x00
         OUT PORTB,R16

         SEI
ind_loop:rjmp ind_loop

int1_ISR:      IN R16,SREG; push the status register into stack
               PUSH R16

               LDI R16,0x0A; variable to blink LED 10 times
               MOV R0,R16
       c2:     LDI R21,15; Loop where LED is ON
               LDI R16,0x01
               OUT PORTB,R16
 ; in each cycle, LED is switched ON for 50% of the time and OFF for   other 50%
       c1:     LDI R17,110
       a1      LDI R20,100
       a2:     DEC R20
               BRNE a2
               DEC R17
               BRNE a1
               DEC R21
               BRNE c1
               LDI R21,15; Loop where LED is OFF
               LDI R16,0x00
               OUT PORTB,R16
       d2:     LDI R17,110
       d3:     LDI R20,100
       d4:     DEC R20
               BRNE d4
               DEC R17
               BRNE d3
               DEC R21
               BRNE d2
               DEC R0
               BRNE c2
               POP R10
               OUT SREG,R10; Pop status register into register R10
               RETI
```

A similar one for the int0 program in assembly code, the only difference being the GICR register is assigned the value 0100 0000 instead, corresponding to int0. Also, the .ORG directive is set to 1 instead of 2 corresponding to int0. The ISR is labelled to int0.

```
rjmp reset

.org 0x0001; interrupt vector for int0 in Atmega 8
rjmp int1_ISR

.org 0x0100

reset:
      LDI R16,0x70; Loading stack pointer address
         OUT SPL,R16
         LDI R16,0x00
         OUT SPH,R16
         LDI R16,0x00
         OUT DDRD,R16
         OUT MCUCR,R16; Set MCUCR register to enable low level interrupt
         LDI R16,0B01000000
         OUT GICR,R16; Set GICR register to 0100 0000 to enable int1
         LDI R16,0x00
         OUT PORTB,R16

         SEI
ind_loop:rjmp ind_loop

int0_ISR:      IN R16,SREG; push the status register into stack
               PUSH R16

               LDI R16,0x0A; variable to blink LED 10 times
               MOV R0,R16
       c2:     LDI R21,15; Loop where LED is ON
               LDI R16,0x01
               OUT PORTB,R16
 ; in each cycle, LED is switched ON for 50% of the time and OFF for   other 50%
       c1:     LDI R17,110
       a1      LDI R20,100
       a2:     DEC R20
               BRNE a2
               DEC R17
               BRNE a1
               DEC R21
               BRNE c1
               LDI R21,15; Loop where LED is OFF
               LDI R16,0x00
               OUT PORTB,R16
       d2:     LDI R17,110
       d3:     LDI R20,100
       d4:     DEC R20
               BRNE d4
               DEC R17
               BRNE d3
               DEC R21
               BRNE d2
               DEC R0
               BRNE c2
               POP R10
               OUT SREG,R10; Pop status register into register R10
               RETI
```

Given below is the flowchart that describes the ISR program. Loops have been implemented in both programs to achieve blinking of LED with 50% duty cycle.

The values of R17, R20 and R21 are chosen such that the entire 3 nested loops together run for 500000 machine cycles. Since R0 = 10, the entire set up will run 10 times.



The C programme to implement the same is given below,

```c
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

ISR (INT0_vect)
{
	int i;
	for (i=1;i<=1;i++) // for 10 times LED blink

	{
		PORTB=0x01;
		_delay_ms(1000);   // delay of 1 sec
		PORTB=0x00;
		_delay_ms(1000);

	}


}
int main(void)
{
	//To enable interrupt and port interfacing
	//For LED to blink
	DDRD=0x02;
	DDRB=0x00; //Make PB0 as output
	MCUCR=0x00; //Set MCUCR to level triggered
	GICR=0x40; //Enable int0
	PORTB=0x00;
	sei();       // global interrupt flag

	while (1) //wait
	{

	}
}
```

A similar code is followed for int1, the ISR definition changes to INT1_vect and the same code with delays is executed. Once the ISR is executed, the programme returns to the while loop and remains there.

```c
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

ISR (INT1_vect)
{
	int i;
	for (i=1;i<=10;i++) // for 10 times LED blink

	{
		PORTB=0x01;
		_delay_ms(1000);   // delay of 1 sec
		PORTB=0x00;
		_delay_ms(1000);

	}


}
```

```
int main(void)
{
	//To enable interrupt and port interfacing
	//For LED to blink
	DDRD=0x03;
	DDRB=0x00; //PB0 as output
	MCUCR=0x00; //Set MCUCR to level triggered
	GICR=0x80; //Enable int1
	PORTB=0x00;
	sei();		// global interrupt flag

	while (1) //wait
	{

	}
}
```

Inference and Conclusion:

The program occupies large amount of program memory in assembly language because we have used loops to solve the issues of delay. The C program is an easier way to implement the same idea because delay concept is implemented in a much simpler fashion through a single line built-in function. So, only the necessary header files are needed to run the code.

In the assembly code, we have direct control over the CPU’s memory and internal registers. But that is not the case in the C code counterpart of the same program.