Instytut Informatyki Politechniki Śląskiej
Zespół Mikroinformatyki i Teorii Automatów Cyfrowych

Laboratorium SMiW

Rok akademicki	Rodzaj studiów*: SSI/NSI/NSM	Numer ćwiczenia:	Grupa	Sekcja
2014/2015	**SSI**	**20 i 21**	**4**	**7**
Data i godzina planowana ćwiczenia: dd/mm/rrrr - gg:mm	**18/11/2014 10:00**	**Prowadzący**: OA/JP/KT/GD/BSz/GB	**JP**	
Data i godzina wykonania ćwiczenia: dd/mm/rrrr - gg:mm	**18/11/2014 10:00**			

Sprawozdanie

Temat ćwiczenia:

Mikrokontrolery z serii AVR

Skład sekcji:	1. Stefania Perlak 2. Wojciech Dudzik 3. Józef Flakus 4. Sebastian Musiał

1. Treść zadania

Do portu A mikrokontrolera AVR Atmega256 o zegarze taktowanym z częstotliwością 10 MHz podłączono 8 przycisków podpiętych do masy. Natomiast do portu B podłączono 8 diod świecących przez rezystor do masy (0 gasi, 1 zapala). W dowolnym miejscu w pamięci programu umieszczono tablice - "tab_ROM" o nieznanej długości i zakończonej słowem 0 (0x0000). Dodatkowo w dowolnym miejscu w pamięci została umieszona tablica o znanej długości "tab_RAM". Program powinien poprawnie działać w przedziale 0-256 bajtów.

2. Polecenie

Naszym zadaniem było napisanie programu, który wyświetli 100 razy zawartość tablicy ROM na diodach. Należało dokonać implementacji w języku C (pierwsza część laboratorium) oraz w Asemblerze AVR (druga część laboratorium), następnie porównać czas wykonywania programu oraz zajętość kodu obu implementacji.

3. Kod programu - C

```
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <util/delay.h>
#define  F_CPU 10000000UL
#define nLength 100

uint8_t TAB_RAM[nLength];

#define GET_FAR_ADDRESS(var) \
({ \
uint_farptr_t tmp; \
__asm__ __volatile__( \
"ldi %A0, lo8(%1)" "\n\t" \
"ldi %B0, hi8(%1)" "\n\t" \
"ldi %C0, hh8(%1)" "\n\t" \
"clr %D0" "\n\t" \
: \
"=d" (tmp) \
: \
"p" (&(var)) \
); \
tmp; \
})


const unsigned char TAB_ROM[12]PROGMEM={0,1,2,3,4,5,6,7,8,9,0x00,0x00};

void main (void)
{
	DDRA=0x00;		// port wejściowy
	PORTA=0x00;		// na wejściu są '0'

	DDRB=0xFF;		// port wyjściowy
	PORTB=0x00;		// na wyjściu zera, żeby nic nie świeciło

	showROM(&TAB_ROM[0]);
}


void showROM(unsigned char*wsk)
{
	unsigned char* tablica=wsk;
	unsigned char x,y;
	int i;
	for (i = 0; i < 100; i++)
	{
```

```
                while(1)
                {
                        x = pgm_read_byte_far(GET_FAR_ADDRESS(tablica));
                        y = pgm_read_byte_far(GET_FAR_ADDRESS(++tablica));
                        if(x==0x00)
                        {
                                if(y==0x00)          // Jeżeli było 0x0000 to kończymy pętlę
                                {
                                        tablica=wsk; )          //ustawienie wskaźnika na początek tablicy
                                        break;
                                }
                        }
                        PORTB=x;
                }
        }
}
```

4. Kod programu – ASM

```
.nolist
.include "m2560def.inc"
.list

;////////////////////////////////////////////////////////////////////////////
; EEPROM - data non volatile memory segment
.ESEG

;////////////////////////////////////////////////////////////////////////////
; StaticRAM - data memory.segment
.DSEG
.ORG 0x100
.EQU RAM_TAB_SIZE = 3              ;definicja wielkości tablicy
RAMTAB: .BYTE RAM_TAB_SIZE         ;definicja tablicy

;////////////////////////////////////////////////////////////////////////////
; CODE - Program memory segment
; Please Remember that it is "word" address space
.CSEG
.org 0x0000
jmp    MAIN                        ;jmp to start of program

jmp    EXT_INT0                    ; IRQ0 Handler
jmp    EXT_INT1                    ; IRQ1 Handler
jmp    EXT_INT2                    ; IRQ2 Handler
jmp    EXT_INT3                    ; IRQ3 Handler
jmp    EXT_INT4                    ; IRQ4 Handler
jmp    EXT_INT5                    ; IRQ5 Handler
jmp    EXT_INT6                    ; IRQ6 Handler
jmp    EXT_INT7                    ; IRQ7 Handler
jmp    TIM2_COMP                   ; Timer2 Compare Handler
jmp    TIM2_OVF                    ;Timer2 Overflow Handler
jmp    TIM1_CAPT                   ;Timer1 Capture Handler
jmp    TIM1_C0MPA                  ;Timer1 CompareA Handler
jmp    TIM1_C0MPB                  ;Timer1 CompareB Handler
jmp    TIM1_0VF                    ;Timer1 Overflow Handler
jmp    TIM0_COMP                   ;Timer0 Compare Handler
jmp    TIM0_OVF                    ;Timer0 Overflow Handler
jmp    SPI_STC                     ;SPI Transfer Complete Handler
jmp    USART0_RXC                  ;USART0 RX Complete Handler
jmp    USART0_DRE                  ;USART0,UDR Empty Handler
jmp    USART0_TXC                  ;USART0 TX Complete Handler
jmp    ADC1                        ;ADC Conversion Complete Handler
jmp    EE_RDY                      ;EEPROM Ready Handler
jmp    ANA_COMP                    ;Analog Comparator Handler
jmp    TIM1_C0MPC                  ;Timer1 CompareC Handler
jmp    TIM3_CAPT                   ;Timer3 Capture Handler
jmp    TIM3_COMPA                  ;Timer3 CompareA Handler
```

```
jmp    TIM3_COMPB                ; Timer3 CompareB Handler
jmp    TIM3_COMPC                ;Timer3 CompareC Handler
jmp    TIM3_OVF                  ;Timer3 Overflow Handler
jmp    USART1_RXC                ;USART1 RX Complete Handler
jmp    USART1_DRE                ;USART1,UDR Empty Handler
jmp    USART1_TXC                ;USART1 TX Complete Handler
jmp    TWI                       ;Two-wire Serial Interface Interrupt Handler
jmp    SPM_RDY                   ;SPM Ready Handler

;////////////////////////////////////////////////////////////////////////////
EXT_INT0:                        ; IRQ0 Handler
EXT_INT1:                        ; IRQ1 Handler
EXT_INT2:                        ; IRQ2 Handler
EXT_INT3:                        ; IRQ3 Handler
EXT_INT4:                        ; IRQ4 Handler
EXT_INT5:                        ; IRQ5 Handler
EXT_INT6:                        ; IRQ6 Handler
EXT_INT7:                        ; IRQ7 Handler
TIM2_COMP:                       ; Timer2 Compare Handler
TIM2_OVF:                        ;Timer2 Overflow Handler
TIM1_CAPT:                       ;Timer1 Capture Handler
TIM1_C0MPA:                      ;Timer1 CompareA Handler
TIM1_C0MPB:                      ;Timer1 CompareB Handler
TIM1_0VF:                        ;Timer1 Overflow Handler
TIM0_COMP:                       ;Timer0 Compare Handler
TIM0_OVF:                        ;Timer0 Overflow Handler
SPI_STC:                         ;SPI Transfer Complete Handler
USART0_RXC:                      ;USART0 RX Complete Handler
USART0_DRE:                      ;USART0,UDR Empty Handler
USART0_TXC:                      ;USART0 TX Complete Handler
ADC1:                            ;ADC Conversion Complete Handler
EE_RDY:                          ;EEPROM Ready Handler
ANA_COMP:                        ;Analog Comparator Handler
TIM1_C0MPC:                      ;Timer1 CompareC Handler
TIM3_CAPT:                       ;Timer3 Capture Handler
TIM3_COMPA:                      ;Timer3 CompareA Handler
TIM3_COMPB:                      ; Timer3 CompareB Handler
TIM3_COMPC:                      ;Timer3 CompareC Handler
TIM3_OVF:                        ;Timer3 Overflow Handler
USART1_RXC:                      ;USART1 RX Complete Handler
USART1_DRE:                      ;USART1,UDR Empty Handler
USART1_TXC:                      ;USART1 TX Complete Handler
TWI:                             ;Two-wire Serial Interface Interrupt Handler
SPM_RDY:                         ;SPM Ready Handler
reti                             ; return from all no used


;////////////////////////////////////////////////////////////////////////////
; Program start
MAIN:
      cli                        ; disable all interrupts
      ldi R16, HIGH(RAMEND)      ; set stack pointer to top of RAM
      out SPH, R16
      ldi R16, LOW(RAMEND)
      out SPL, R16

;----------------------------------------------------------------------------
; Main program code place here
; 1. Place here code related to initialization of ports and interrupts

      ldi r16, 0x00              ;wartość do ustawiania portów na 0
      ldi r17, 0xFF              ;wartość do ustawiania portów na 1
      out DDRB, r17              ;port wyjściowy
      out PORTB, r16             ;na wyjściu zera
      out DDRA, r16              ;port wejściowy
      out PORTA, r16             ;na wejściu są '0'

;----------------------------------------------------------------------------
; F2. Load initial values of index registers
```

```
;  Z, X, Y

LDI r20, 0x00                    ;inicjalizacja licznika początku pętli

for_loop:
       ldi r30, byte1(ROM_TAB<<1) ;ładowanie adresu w rom do rejestru Z
       ldi r31, byte2(ROM_TAB<<1) ;ładowanie adresu w rom do rejestru Z
       ldi r16, byte3(ROM_TAB<<1) ;instrukcje te służą możliwości czytania romu w dowolnym
                                   miejscu w pamięci
       out RAMPZ, r16

       cpi r20, 0x64               ;porównanie zmiennej sterującej pętla z 100
       breq end_for                ;jeśli równe to koniec pętli

loop:                              ;wczytaj z Rom TAB
       elpm r16, Z+                ;wczytaj wartość i zwiększ adres
       elpm r17, Z                 ;wczytaj kolejna wartość

                                   ;sprawdź, czy koniec
       cpi r16, 0x00               ;sprawdzenie czy 0 (początek strażnika)
       brne non_0000               ;skok względy gdy nie są równe
       cpi r17, 0x00               ;sprawdzenie czy 0 (druga cześć strażnika)
       breq display_end            ;jeśli tak to osiągnięto koniec tablicy
       non_0000:                   ;etykieta gdy nie koniec

       out PORTB, r16              ;wyświetl rom_tab[i]
       nop
       rjmp loop

display_end:                       ;koniec jednokrotnego wyświetlenia zawartości tablicy
       inc r20
       rjmp for_loop

;-----------------------------------------------------------------------------
; Program end - Ending loop
;-----------------------------------------------------------------------------
end_for:
       rjmp END


.ORG 0x17FFE                       ;przesuwamy sie w rozne miejsca pamieci
;-----------------------------------------------------------------------------
; Table Declaration -  place here test values
; Test with different table values and different begin addresses of table (als above 0x8000)
;
ROM_TAB: .DB 0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x09,0x00,0x00
.EXIT
;-----------------------------------------------------------------------------
```

5. Uruchamianie i testowanie

Programy były uruchamiane i testowane w symulatorze programu Atmel Studio 6. Dodatkowo aby sprawdzić poprawność działania kodu programów, tablica tab_ROM była przez nas umieszczana w różnych miejscach pamięci mikrokontrolera(asembler). Przykładowe adres pod jakimi umieściliśmy tablicę to:

- 0x7000
- 0x8000
- 0x17FFE
- 0xFFA0

6. Porównanie pamięciowe

Rozmiar tablicy	Rozmiar w C (bajty)	Rozmiar w ASM (bajty)
10	376	223
20	386	233
30	396	243
40	406	253
50	416	263
60	426	273

Jak widać rozmiar rósł proporcjonalnie do rozmiaru tablicy (tablica zawierała elementy bajtowe).

7. Porównanie czasowe

Rozmiar tablicy	Czas w C (μs)	Czas w ASM (μs)
10	2 626,90	1 292,20
20	5 424,90	2 592,20
30	8 224,90	3 892,20
40	11 024,90	5 192,20
50	13 824,90	6 492,20
60	16 624,90	7 792,20

Czas wykonania programu o identycznej funkcjonalności w asemblerze był średnio 2 razy szybszy niż jego odpowiednik z języka C.

8. Wnioski

Celem laboratorium było zapoznanie się z programowaniem mikrokontrolerów AVR. Zadanie umożliwiało pisanie w języku C i w asemblerze, tak by można było porównać oba sposoby programowania. Mimo faktu, że implementowano ten sam algorytm w obu językach, wyniki znacznie się różnią, zarówno jeśli chodzi o czas, jak i o zajętość pamięci. Uzyskane wyniki przedstawione w powyższych tabelach pokazują wyższość asemblera nad językiem wyższego poziomu (w tym przypadku języka C). Ponad dwa razy krótszy czas wykonywania programu jest bardzo dobrym wyznacznikiem przewagi programowania niskopoziomowego, tym bardziej, jeśli weźmiemy pod uwagę niskie skomplikowanie naszego programu.
Rozważając zalety i wady obu metod nie należy jednak zapominać o większym nakładzie czasowym położonym na programowanie w asemblerze. Należy również wziąć pod uwagę większe wymaganie co do umiejętności.
Kwestią sporną pozostaje zatem czy pisać szybko wolniejsze programy, czy wolniej – szybsze. Uwagę należy również zwrócić na zajętość pamięciową programu w tym aspekcie również asembler jest lepszy a w przypadku rozbudowanych układów, gdzie wykorzystuje się zewnętrzne biblioteki dla sterowania ekranem, lub układów z niewielką ilością pamięci na program (np.: seria AVR aTiny) mniejsza zajętość pamięciowa może okazać się największą zaletą. Dlatego należy się zastanowić które z narzędzi okaże się dla nas lepsze w danej sytuacji.