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, 21**	**1**	**2**

Data i godzina planowana ćwiczenia: dd/mm/rrrr - gg:mm	**03.11.2014 10:00** **24.11.2014 10:00**	**Prowadzący**: OA/JP/KT/GD/BSz/GB	**JP**
Data i godzina wykonania ćwiczenia: dd/mm/rrrr - gg:mm	**03.11.2014 10:00** **24.11.2014 10:00**		

Sprawozdanie

Temat ćwiczenia:

Mikrokontrolery z serii AVR

Skład sekcji:	1. Aleksander Grzybowski 2. Martin Białkowski 3. Marek Mrowiec 4. Alicja Zorzycka

1. Treść zadania.

Założenia

ATMEGA 2560, 16 MHz.
port A - 8 przycisków do masy
port B - 8 diod świecących przez rezystor do masy
1 zapala, 0 gasi.
Gdzieś w pamięci kodu jest tablica TAB_ROM o nieznanej długości zakończona "zero". 2B
Jest również TAB_RAM
0-512 bajtów

Treść

Po naciśnięciu dowolnego przycisku (które podpięte są pod PORTA) przekopiować zawartość tablicy nr 1 (znajdującej się w pamięci ROM) do tablicy nr 2 (znajdującej się w pamięci RAM). Tablica ROM zakończona jest strażnikiem 0x0000, strażnik nie jest kopiowany.

Dodatkowe informacje oraz wskazówki i wnioski z laboratorium.

Program w języku C został zabezpieczony przed podaniem nieprawidłowych długości tablic poprzez dyrektywę preprocesora, natomiast asemblerowy poprzez sprawdzanie tych długości w runtimie.
Nie zostało powiedziane, jak powinien zachować się program po dokonaniu kopiowania, więc zakładam, że program kończy swoje działanie. Nie jest to sprzeczne z założeniami zadania (poza tym bez sensu jest kopiować to samo drugi raz), a znacząco upraszcza debugowanie (nie trzeba odklikiwać przycisku za każdym razem)

2. Kod w C.

```
#define F_CPU 16000000L
#include <avr/io.h>
#include <avr/eeprom.h>
#include <avr/pgmspace.h>

#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;                                              \
})

#define ROMTAB_SIZE 514 // 2=strażnik
#define RAMTAB_SIZE 512
unsigned const char PROGMEM romtab[ROMTAB_SIZE] =
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,3
```

```
3,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63
,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,
94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,1
18,119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,137,138,139,140,
141,142,143,144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,163
,164,165,166,167,168,169,170,171,172,173,174,175,176,177,178,179,180,181,182,183,184,185,18
6,187,188,189,190,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208,2
09,210,211,212,213,214,215,216,217,218,219,220,221,222,223,224,225,226,227,228,229,230,231,
232,233,234,235,236,237,238,239,240,241,242,243,244,245,246,247,248,249,250,251,252,253,254
,255,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,
32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,6
2,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92
,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,1
17,118,119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,137,138,139,
140,141,142,143,144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162
,163,164,165,166,167,168,169,170,171,172,173,174,175,176,177,178,179,180,181,182,183,184,18
5,186,187,188,189,190,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,2
08,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,224,225,226,227,228,229,230,
231,232,233,234,235,236,237,238,239,240,241,242,243,244,245,246,247,248,249,250,251,252,253
,254,255, 0, 0};
char ramtab[RAMTAB_SIZE];

#if RAMTAB_SIZE < (ROMTAB_SIZE-2)
#error "Zbyt mala tablica w pamieci RAM"
#endif

// po naciśnięciu przycisku (na PORTA) przekopiuj z romu do ramu

int main(void)
{

	DDRA = 0x00; // 8 wejść
	PORTA = 0xFF; // + pullup-y

	// czekaj na naciśnięcie (na jakąś masę na PA, nic nie wciśnięte = 0xff)
	while (PINA == 0xff);

	int offset = 0;
	while (1) {
		char first = pgm_read_byte_far(GET_FAR_ADDRESS(romtab) + offset);
		char second = pgm_read_byte_far(GET_FAR_ADDRESS(romtab) + offset + 1);
		if (first == 0x00 && second == 0x00) break;
		ramtab[offset] = first;
		offset++;
	}

	// po skopiowaniu zakończ działanie
	while(1);
}
```

3. Kod w Asemblerze.

```
;/////////////////////////////////////////////////////////////////////////
; Laboratory AVR Microcontrollers Part1
; Program template for lab 20
; Authors:
;
; Group:
; Section:
;
; Task:
;
; Todo:
;
```

```
;
; Version: 1.0
;/////////////////////////////////////////////////////////////////////////
.nolist
.include "m128def.inc"
.list

.equ RAMTAB_SIZE=512
.equ ROMTAB_SIZE=514

;/////////////////////////////////////////////////////////////////////////
; StaticRAM - data memory.segment
.DSEG
.ORG 0x100; may be omitted this is default value
; Destination table (xlengthx bytes).
; Replace "xlengthx" with correct value
ramtab: .BYTE RAMTAB_SIZE

;/////////////////////////////////////////////////////////////////////////
; CODE - Program memory segment
; Please Remember that it is "word" address space
;
.CSEG
.org 0x0000 ; may be omitted this is default value
jmp    RESET  ; Reset Handler

; Interrupts vector table / change to your procedure only when needed
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
RESET:

cli          ; disable all interrupts

; sprawdź, czy można kopiować, trzeba porównać liczby większe od 255 (tj. <0,512>) więc na
2 razy
; najpierw starszy bajt możliwie 2-bajtowej liczby
ldi r16, high(RAMTAB_SIZE)
cpi r16, high(ROMTAB_SIZE-2)
brlo koniec
ldi r16, low(RAMTAB_SIZE)
cpi r16, low(ROMTAB_SIZE-2)
brlo koniec



;  Set stack pointer to top of RAM
ldi R16, HIGH(RAMEND)
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

; porty
ldi r16, 0x00
out DDRA, r16
ldi r16, 0xff
out PORTA, r16
```

```
; co prawda nie używamy diod, ale...
ldi r16, 0x00 ; stan wysokiej imped - diody nie świecą
out DDRB, r16



; czekaj na guzik
czekaj:
in r16, PINA
cpi r16, 0xff
breq czekaj

; RAMPZ dla rom żeby obejść 64kB limit
ldi r30, low(romtab<<1)
ldi r31, high(romtab<<1)
ldi r16, byte3(romtab<<1)
out RAMPZ, r16

; przygotuj czytanie do pamieci RAM
ldi r28, low(ramtab) ;
ldi r29, high(ramtab) ;

 loop:
; wczytaj 2 bajty z romtab
elpm r16, Z+
elpm r17, Z
; if (t[0] == 0 && t[1] == 0) ...
cpi r16, 0x00
brne kopiuj
cpi r17, 0x00
brne kopiuj

koniec:
rjmp koniec

kopiuj:
; tu kopiowanie

st Y+,r16 ; kopiuj bajt

rjmp loop

;---------------------------------------------------------------------------
; Table Declaration -  place here test values
; Test with different table values and different begin addresses of table (als above
0x8000)
;
.org 0x7ff0
romtab: .DB
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33
,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,
64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,9
4,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,11
8,119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,137,138,139,140,1
41,142,143,144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,
164,165,166,167,168,169,170,171,172,173,174,175,176,177,178,179,180,181,182,183,184,185,186
,187,188,189,190,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208,20
9,210,211,212,213,214,215,216,217,218,219,220,221,222,223,224,225,226,227,228,229,230,231,2
32,233,234,235,236,237,238,239,240,241,242,243,244,245,246,247,248,249,250,251,252,253,254,
255,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,3
2,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62
,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,
93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,11
7,118,119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,137,138,139,1
40,141,142,143,144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,
163,164,165,166,167,168,169,170,171,172,173,174,175,176,177,178,179,180,181,182,183,184,185
```

```
,186,187,188,189,190,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,20
8,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,224,225,226,227,228,229,230,2
31,232,233,234,235,236,237,238,239,240,241,242,243,244,245,246,247,248,249,250,251,252,253,
254,255, 0, 0
.EXIT
;-----------------------------------------------------------------------
```

4. Uruchamianie i testowanie.

Symulacje przeprowadzone zostały w środowisku Atmel Studio 6.2. Zgodnie z sugestią prowadzącego zweryfikowałem zachowanie programu asemblerowego, w przypadku, gdy tablica ROM znajduje się odpowiednio: poza blokiem 64k (org 0x8100) oraz na "zakładce" (org 0x7ff0). Dla tych przypadków program zachowuje się prawidłowo.
Sprawdzenia poprawności wykonanego programu dokonywałem, korzystając z podglądu pamięci. Jako iż za dane testowe wybrałem kolejne liczby naturalne (0-255), mogłem w krótkim czasie sprawdzić, czy kopiowanie się powiodło.

5. Porównanie kodów.

Pomiarów dokonałem korzystając z funkcji stopwatch okienka Processor, używając kwarcu 16M i pomijając czas inicjalizacji który występuje w języku C. W celu ułatwienia porównań, użyłem największego możliwego zbioru danych testowych, to jest tablica 512 bajtów.

	C	ASM
Rozmiar w bajtach	878	210
Czas wykonania [µs]	1315	418,5

6. Wnioski.

Jak można zauważyć, program napisany w asemblerze jest zarówno szybszy, jak i mniejszy. Warto więc stosować go na tiniaczach i podobnych, słabych acz tanich mikrokontrolerach AVR, ponieważ można wtedy wycisnąć maksimum mocy z takiego układu i zaoszczędzić pieniądze. Z drugiej strony, tworzenie wsadów w C jest znacznie wygodniejsze, składnia pozwala znacząco uprościć skomplikowane operacje i wymusić np. poprawne nagłówki funkcji. Należy zawsze zastanowić się nad wyborem języka, ponieważ zdeterminuje on dalsze prace nad projektem.