<table>
<tr><td></td><td colspan="3">Instytut Informatyki Politechniki Śląskiej
Zespół Mikroinformatyki i Teorii Automatów Cyfrowych
Laboratorium SMiW</td><td></td></tr>
<tr><td>Rok akademicki</td><td>Rodzaj studiów*: SSI/NSI/NSM</td><td>Numer ćwiczenia:</td><td>Grupa</td><td>Sekcja</td></tr>
<tr><td>2015/2016</td><td>SSI</td><td>7</td><td>4</td><td>2</td></tr>
<tr><td>Data i godzina planowana ćwiczenia:
dd/mm/rrrr - gg:mm</td><td>20/10/2015 8:30</td><td rowspan="2">Prowadzący:
JP/KT/GD/GB</td><td colspan="2" rowspan="2">JP</td></tr>
<tr><td>Data i godzina wykonania ćwiczenia:
dd/mm/rrrr - gg:mm</td><td>20/10/2015 8:30</td></tr>
<tr><td colspan="5">Sprawozdanie</td></tr>
<tr><td colspan="5">Temat ćwiczenia:

Mikrokontrolery z serii AVR</td></tr>
<tr><td>Skład sekcji:</td><td colspan="4">1. Dominika Błasiak
2. Aleksandra Chronowska</td></tr>
</table>

1. Treść zadania

Do portu A mikrokontrolera AVR Atmega256 o zegarze taktowanym z częstotliwością 4 MHz podłączono 8 przycisków podpiętych do masy.
Do portu B podłączono 8 diod świecących przez tranzystor do zasilania.
W dowolnym miejscu w pamięci programu umieszczono tablice - "tab_ROM" o nieznanej długości i zakończonej słowem 0 x FF oraz tablice o znanej długości "tab_RAM".
Program powinien poprawnie działać w przedziale 0-1024 bajtów.
Program ma wyświetlać zawartość tablicy" tab_ROM" za pomocą diod po naciśnięciu pojedynczego przycisku.
Założenie: Po kliknięciu dowolnego przycisku na diodach zostaje wyświetlony pierwszy bajt tablicy. Wczytanie i wyświetlenie kolejnej wartości tablicy odbywa się po puszczeniu danego przycisku i ponownym kliknięciu dowolnie wybranego przycisku (kliknięcie przycisku -> wyświetlenie na diodach -> puszczenie przycisku -> kliknięcie przycisku -> wyświetlenie na diodach -> puszczenie przycisku itd. dopóki program nie napotka 0xFF). Program kończy działanie, gdy napotka 0xFF.

2. Kod programu – język wysokiego poziomu C

```
3.  #include <avr/io.h>
4.  #include <avr/pgmspace.h>
5.
6.  #define ROM_SIZE 5
7.  #define F_CPU 4000000UL //clock = 4MHz
8.  //tablica umieszczona gdzieś w pamięci ROM
9.  const char ROMTAB[ROM_SIZE] PROGMEM= {1,2,3,4,0xFF};
10.
11. //inicjalizacja portów
12. void initPorts()
13. {
14.         DDRA = 0x00;    //port a jako wejscie przez rezystor podciagajacy
    (podłączamy do niego klawisze)
15.         PORTA = 0xFF; //PORTA podpinamy do masy
16.         DDRB = 0xFF;    //port b jako wyjscie (podłączamy do niego diody)
17.         PORTB = 0xFF;//zapewniamy napiecie wysokie, aby diody nie swiecily
18. }
19.
20. void printROMTAB()
21. {
22.         int i=0;
23.         //wczytaj pierwszy element z tablicy
24.         unsigned char dataBit = pgm_read_byte(&(ROMTAB));
25.         while(1)
26.         {
27.                 //czeka na wciśnięcie klawisza
28.                 while(PINA == 0xFF);
29.                 //zapalenie wybranej diody odbywa sie przez wystawienie 0 na
    odpowiednim pinie portu B
30.                 PORTB = (dataBit) ^ 0xFF; //0 xor 1 = 1 ; 1 xor 1 = 0;
31.                 // jeśli wczytany element jest równy 0xFF wtedy kończy się
    wykonywanie pętli
32.                 if(dataBit == 0xFF)
33.                 break;
34.                 i++;
```

```
35.             //wczytanie kolejnego elementu
36.             dataBit = pgm_read_byte(&(ROMTAB[i]));
37.             //czekanie na puszczenie klawisza
38.             while(PINA != 0xFF);
39.         }
40. }
41. int main(void)
42. {
43.         initPorts();
44.         printROMTAB();
45. }
```

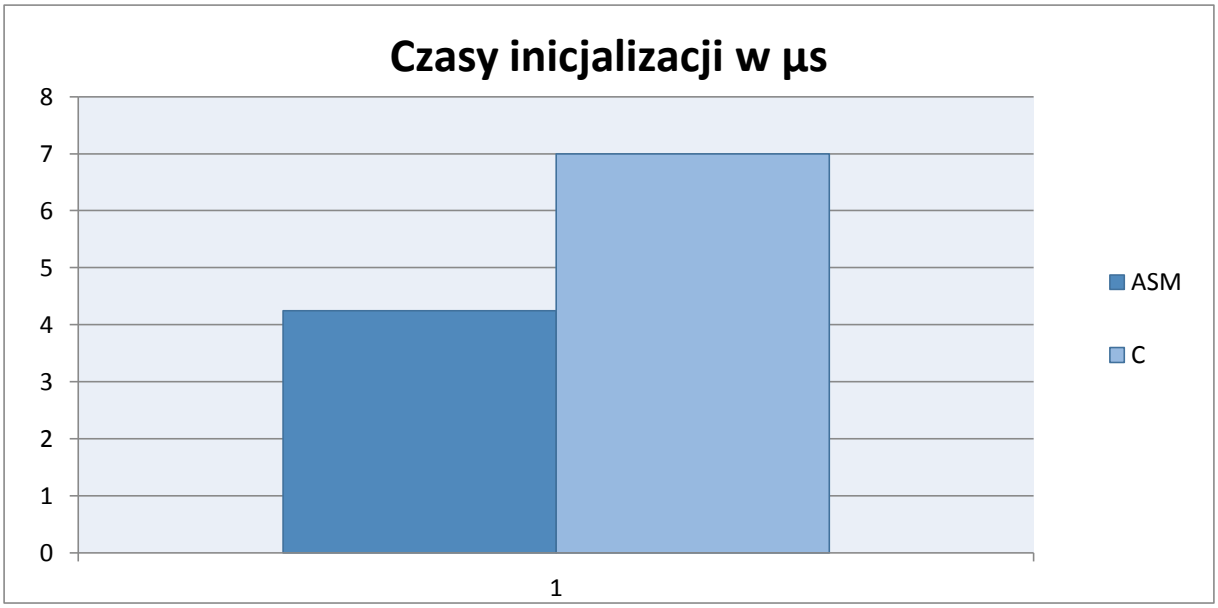
3. Kod programu – język niskiego poziomu ASM

```
4.  .nolist
5.  .include "m2560def.inc"
6.  .list
7.  .ESEG ; EEPROM memory segment
8.  .DSEG ; SRAM memory.segment
9.  .ORG 0x100; may be omitted this is default value
10. .CSEG ; CODE Program memory. Remember that it is "word" address space
11. .org 0x0000
12. jmp RESET ; Reset Handler
13. RESET:
14. ldi r20, 0xFF ; Maska.
15. ldi r16, byte3(ROMTAB<<1) ; Wskazanie odpowiedniego bloku pamięci (byte3() -
    trzeci bajt).
16. out RAMPZ, r16 ; Wpisanie adresu do rejestru RAMPZ.
17. ; RAMPZ (RAM Page Z Select Register) Rejestr ten określa numer strony pamięci
    programu.
18. ldi ZH, high(ROMTAB<<1) ; Załadowanie starszej części adresu do rejestru Z.
19. ldi ZL, low(ROMTAB<<1) ; Załadowanie młodszej części adresu do rejestru Z.
20. ; Wskaźnik Z potrafi zaadresować jedynie 64K pamięci.
21.
22. cli ; Wyłączenie obsługi przerwań.
23.
24. ; Inicjalizacja portów.
25. ; Port A
26. ldi  r16, 0x00 ;
27. out  DDRA,r16 ; Ustawienie portu A jako wejście.
28. ldi  r16,0xFF ;
29. out  PORTA,r16 ; Wejścia typu PULL-UP.
30. ; Port B
31. ldi r16, 0xFF ;
32. out DDRB, r16 ; Ustawienie portu B jako wyjście.
33. ldi r16, 0xFF ;
34. out PORTB, r16 ; Wygaszenie diod (port B w stanie wysokim).
35. ; Koniec inicjalizacji portów.
36.
37. sei ; Włączenie obsługi przerwań.
38. ; Oczekiwanie na wciśnięty przycisk (dowolny).
39. ; r16 - aktualny stan portu B.
40. Czekanie_na_przycisk:
41. in r16, PINA ; Wczytanie stanu pinów portu A do rejestru r16.
42. cpi r16, 0xFF ; Sprawdzamy czy wciśnięty został jakiś przycisk.
43. breq Czekanie_na_przycisk ; Gdy nie, czekamy na wciśnięty przycisk.
44.
45. elpm  R17, z
46. ; Odczyt kolejnych wartości z ROMTAB.
```

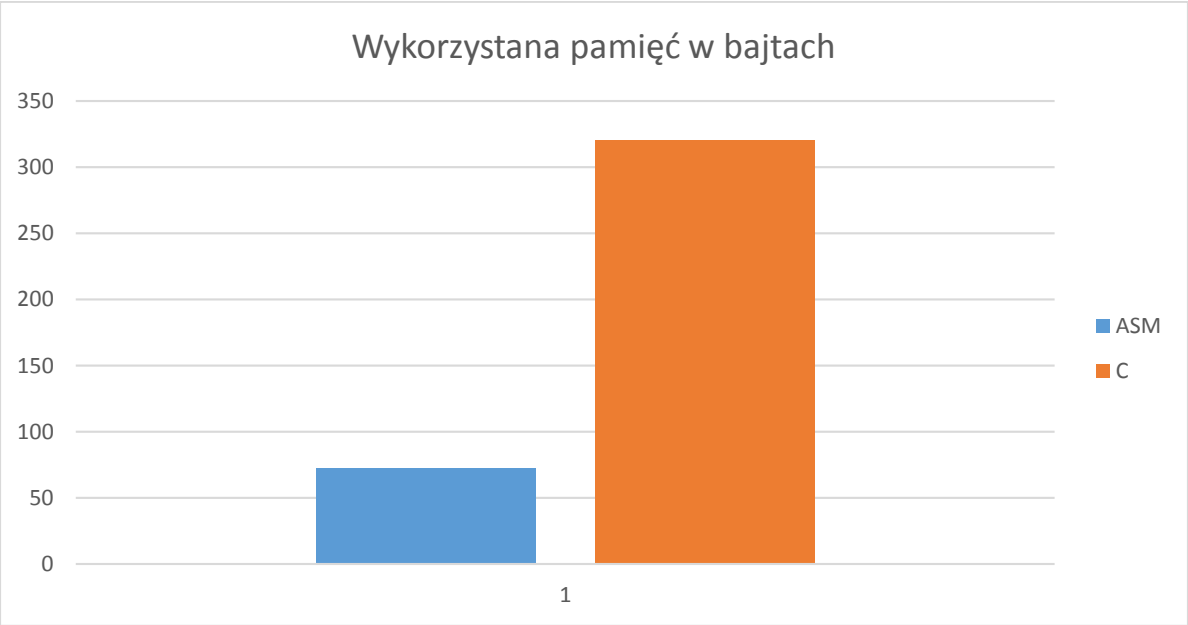
```
47. ; r16 - odczytana wartość tablicy.
48. Odczyt:
49. elpm r16, z+ ; Odczyt wartości oraz inkrementacja Z.
50. ; Wykorzystanie adresowania RAMPZ, ZH ,ZL wymaga użycia instrukcji
51. elpm.
52. cpi r16, 0xFF ; Sprawdź, czy wczytany element to FFh.
53. breq Wyswietl_ostatni_bajt     ; Gdy tak, skocz do end (wyświetli zawartośc
    ostatniego bajta i zakończy program)
54. eor r16, r20
55. out PORTB, r16 ; Gdy nie, wyświetl wartość na diodach.
56. Czekanie_na_odklikniecie:
57. in r16, PINA
58. cpi r16,0xFF
59. brne Czekanie_na_odklikniecie
60. rjmp Czekanie_na_przycisk
61. Wyswietl_ostatni_bajt: ;wyświetlenie zawartości ostatniego bajta na diodach,
    czyli 0xFF
62. eor r16, r20
63. out PORTB, r16 ; Wyświetlenie 0xFF na diodach
64. ; Zakończenie programu.
65. End:
66. rjmp END
67. ; .org 0x8000 -> "wyrównanie słowowe" (WORD-aligned) = 0x10000 -> "wyrównanie
    bajtowe" (BYTE-aligned).
68. .org 0x8000
69. ROMTAB: .db 0x01, 0x02, 0x03, 0x04, 0x05, 0x00, 0x06, 0xFF
70. .EXIT
```

4. Porównanie czasowe i pamięciowe kodu

Porównywanie czasów inicjalizacji:



Porównanie zajętości pamięci:



5. Wnioski

Laboratorium miało na celu zapoznanie z programowaniem mikrokontrolerów z rodziny AVR. Na zajęciach naszym zadaniem było napisanie kodu programu w środowisku Atmel Studio zarówno w języku wysokiego poziomu – C jak i w języku asemblerowym.

Niestety w naszym przypadku zmierzenie czasu wykonania programu było bardzo trudne. Czas zależał głównie od tego jak szybko użytkownik kliknie bądź odkliknie wybrany przycisk, dlatego nasze pomiary nie były wiarygodne. Mimo wszystko podczas testowania można było zauważyć, że czas wykonywania poleceń w asemblerze jest znacznie krótszy niż w C.

Język asemblerowy wykazał się również optymalniejszy w przypadku wykorzystywania pamięci.

Można stwierdzić, że jakość kodu napisanego językiem niskopoziomowym wymaga obszerniejszego zakresu umiejętności i większego wysiłku od programisty.

Jednak program w języku wysokiego poziomu jest znacznie czytelniejszy, łatwiejszy do zaimplementowania. Dodatkową zaletą są również, możliwe do wykorzystania, gotowe biblioteki.