Termin zajęć Czwartek 13:15-15:00	**Podstawy Techniki Mikroprocesorowej**	
Osoby wykonujące ćwiczenie: Michał Bernacki-Janson, Adam Czekalski		**Grupa nr:** **D**
Tytuł ćwiczenia: Arytmetyka, logika, pomięć, diody i brzęczyki		**Ćwiczenie nr:** **1**
Data wykonania ćwiczenia	16.03.2023	**Ocena:**
Data oddania sprawozdania	26.03.2023	

1. Wstęp

Celem pierwszych zajęć było zapoznanie się z działaniem mikrokomputera 8051. Wykonano programy w języku assembler wykonujące: podstawowe operacje arytmetyczne z wyświetleniem wyników na diodach, sortowanie bąbelkowe, wyświetlanie sekwencji na diodach i program, który obsługiwał brzęczyk.

2. Program wykonujący podstawowe operacje arytmetyczne na dwóch argumentach 8-bitowych i dwóch argumentach 16-bitowych, prezentacja wyników na diodach.

Kod z komentarzem:

```
ljmp start
org 0100h

    start:
    ;dodawanie
    mov P1, #00h     ;wyzerowanie diod
    mov a, #01h      ;wczytanie pierwszej liczby do akumulatora
    mov r0, #02h     ;wczytanie drugiej liczby do rejestru
    add a, r0        ;wykonanie dodawania
    clr c            ;wyczyszczenie flagi przeniesienia (carry)
    xrl a, #0ffh     ;zamiana zer na jedynki (0 - dioda zapalona)
    mov P1, a        ;przeniesienie wyniku dodawania do rejestru diod

    ;wyzerowanie diod
    mov P1, #00h

    ;odejmowanie
    mov a, #05h
    mov r1, #0ah
    subb a, r1       ;wykonanie odejmowania, operacja subb uwzglednia "pozyczke" z flagi c, dlatego ja wyzej czyscimy
    xrl a, #0ffh
    mov P1, a

    ;wyzerowanie diod
    mov P1, #00h

    ;mnozenie
    mov a, #05h
    mov b, #04h      ;wczytanie drugiej liczby do rejestru pomocniczego
    mul ab           ;wykonanie mnozenia -> wynik zajmuje 16 bitow wiec trafia do obu rejestrow wejsciowych
    xrl a, #0ffh
    mov P1, a
    xrl b, #0ffh
    mov P1, b

    ;wyzerowanie diod
    mov P1, #00h

    ;dzielenie
    mov a, #0fh
    mov b, #03h
    div ab           ;wykonanie dzielenia -> wynik trafia do akumulatora, a reszta do rejestru pomocniczego
    xrl a, #0ffh
    mov P1, a
    xrl b, #0ffh
    mov P1, b

    ;wyzerowanie diod
    mov P1, #00h

    ;dodawanie dwoch liczb 16-bitowych (o szerokosci dwoch rejestrow)
    mov r0, #01h     ;/*
    mov r1, #02h     ;   wczytanie danych do rejestrow
    mov r2, #03h     ;   zapis liczb: pierwsza: r0, r1, druga: r2, r3
    mov r3, #04h     ;*/

    mov a, r1        ;przeniesienie zawartosci rejestru r1 - do akumulatora
    add a, r3        ;dodawanie bitow nizszej wagi
    mov r4, a
    mov a, r0
    addc a, r2       ;dodawanie bitow wyzszej wagi z uwzglednieniem przeniesienia
    mov b, r4
    xrl a, #0ffh
    mov P1, a        ;wyswietlamy wynik od najbardziej znaczacego bitu
    xrl b, #0ffh
    mov P1, b

    nop
    nop
    jmp $
    END start
```

Wykonujemy dodawanie, odejmowanie, mnożenie, dzielenie oraz dodawanie dwóch liczb 16-bitowych zapisanych na dwóch rejestrach. Na diodach wyświetlane są wyniki każdej z operacji arytmetycznej.

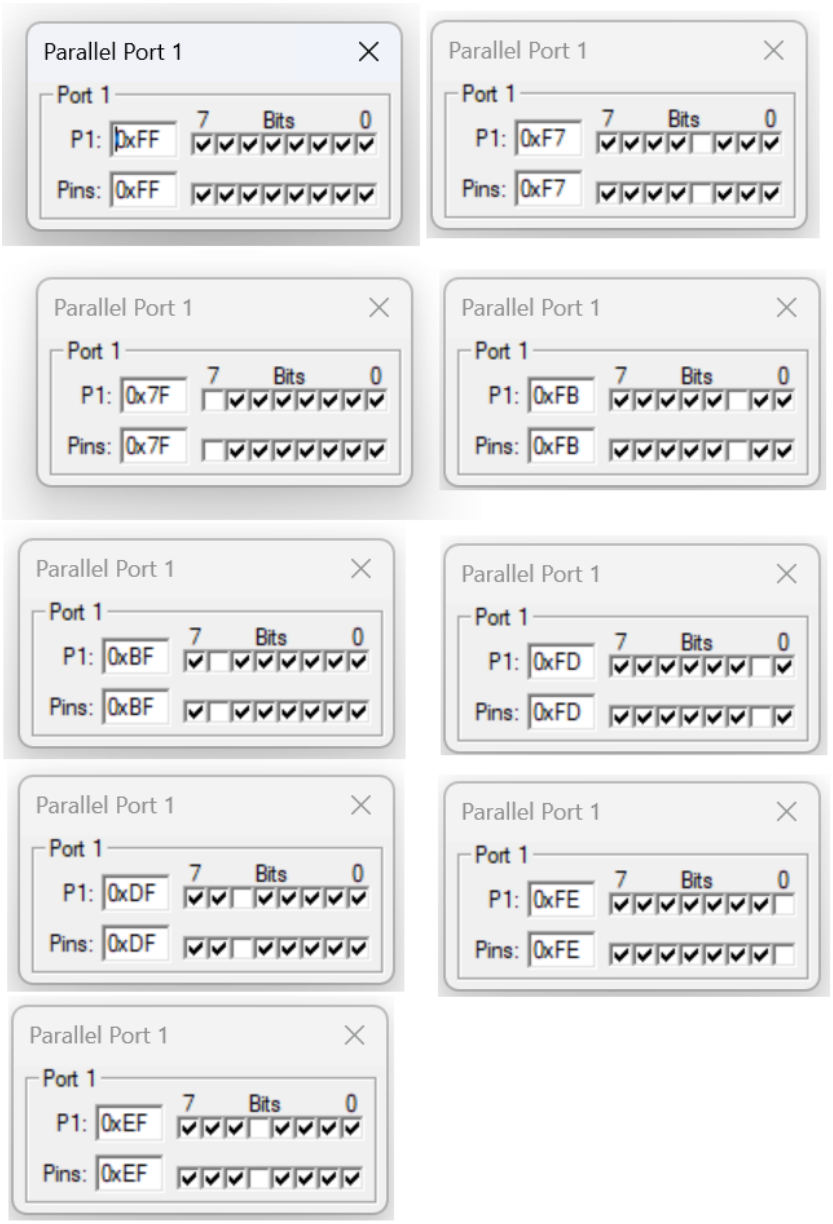
3. Program realizujący ciekawe zapalanie/gaszenie diod podłączonych do portu P1.

Kod z komentarzem:

```
1   ljmp start
2   org 0100h
3
4       ;petla tworzaca opoznienie
5       delay:  mov r0, #0FFH
6       one:    mov r1, #0FAH
7       dwa:    djnz r1, dwa
8           djnz r0, one
9           ret
10
11      ;sekwecja zapalania sie diod zapisana w kodzie binarnym
12      ;(0 - dioda zapalona, 1 - dioda zgaszona)
13      start:
14      mov p1, #01111111b
15      lcall delay
16      mov p1, #10111111b
17      lcall delay
18      mov p1, #11011111b
19      lcall delay
20      mov p1, #11101111b
21      lcall delay
22      mov p1, #11110111b
23      lcall delay
24      mov p1, #11111011b
25      lcall delay
26      mov p1, #11111101b
27      lcall delay
28      mov p1, #11111110b
29      lcall delay
30      lcall start
31      END
```

W programie wykorzystano „pętlę” tworzącą opóźnienie. Jej działanie opiera się na tym, że do rejestru r0 wpisujemy wartość FF, następnie do rejestru r1 wpisujemy wartość FA, a potem, dopóki wartość przechowywana w rejestrze r1 nie będzie równa 0, dekrementujemy ją. Następnie dekrementujemy wartość w rejestrze r0 i znów ładujemy wartość FA do rejestru r1 i powtarzamy cały ten proces, dopóki wartość w rejestrze r0 nie będzie równa 0. „Pętlę” wywołujemy po każdym zapaleniu sekwencji diod.

Tak wygląda sekwencja w praktyce:



4. Program wykonujący dźwięk za pomocą brzęczyka podłączonego do portu P3.2

```
ljmp start
org 0100h
    delay:    mov r0, #0FFH ; opoznienie
    one:    mov r1, #0AH
    dwa:    djnz r1, dwa ; zmniejszanie wartosci rejestrow
        djnz r0, one
        ret
    start:
    cpl p3.2 ; zanegowanie bitu 2 w porcie P3
    lcall delay ; wywolanie opoznienia
    lcall start ; powrot na start
    END
```

Tak jak w przypadku programu z sekwencją diod, tutaj też wykorzystujemy pętlę tworzącą opóźnienie w celu spowolnienia zmiany stanu brzęczyka. Na początku zastosowano większe opóźnienie (przy etykiecie „one” do rejestru r1 przenoszono większą wartość), co za tym idzie zmniejszono częstotliwość, w efekcie czego brzęczyk stukał. Następnie zmniejszono opóźnienie w programie, co spowodowało wydobycie się z brzęczyka dźwięku o większej częstotliwości.

Dźwięk z brzęczyka został wydobyty poprzez negację bitu 2 w porcie P3.

5. Program realizujący sortowanie bąbelkowe lub znajdujący minimum albo maksimum tablicy 1-wymiarowej rozpoczynającej się od adresu 8000H pamięci zewnętrznej danych (XRAM) i obejmującej 16 kolejnych komórek tej tablicy.

Kod z komentarzem:

```
1  ljmp start
2  org 8000h
3      dane: db 15,14,13,12,11,10,09,08,07,06,05,04,03,02,01,00
4
5  org 0100h
6
7  fswap:              ;funkcja zamieniajaca miejscami liczby
8      mov a, r0
9      movx @dptr, a
10     dec dpl         ;zmniejszmy polowe mniejszej wagi dptr-a o 1
11     mov a, r1
12     movx @dptr, a
13     inc dpl
14     acall powrot
15
16 start:
17     mov dptr, #dane
18     mov r7, #00h    ;aktualny wskaznik pamieci xdata
19     mov r6, dpl     ;r6 - dpl poczatkowy
20 ladowanie:          ;ladujemy dane z code memory do xdata memory
21     mov a, r7
22     movc a, @a+dptr
23     mov b, a
24     mov a, r7
25     add a, dpl
26     mov dpl, a
27     mov a, b
28     movx @dptr, a
29     mov dpl, r6
30     inc r7
31     cjne r7, #010h, ladowanie
32
33     mov r5, #00h    ;wskaznik tablicy zewnetrznej
34 druga:
35     mov b, #00h     ;wskaznik na element tablicy wewnetrznej
36 pierwsza:
37     ;ladowanie dwoch liczb
38     inc b
39     movx a, @dptr   ;pobieramy pierwsza liczbe do porownania - r0
40     mov r0, a
41     inc dptr        ;zwiekszamy pointer na adres w xdata
42     movx a, @dptr   ;pobieramy druga liczbe do porownania - r1
43     mov r1, a
44     ;porownanie poprzez odjecie i sprawdzenie flagi przeniesienia
45     mov a, r1
46     clr c
47     subb a, r0
48     mov a,#00h
49     jc fswap
50 powrot:
51     mov a, b
52     add a, r5       ;optymalizacja algorytmu
53     cjne a, #0fh, pierwsza
54     mov dpl, r6
55     inc r5
56     cjne r5, #0fh, druga
57
58     mov r5, #00h
59     mov dpl, r6
60 wyswietl:
61     inc r5
62     movx a, @dptr
63     inc dpl
64     cjne r5, #010h, wyswietl
65
66
67     nop
68     nop
69     nop
70     jmp $
71     end start
72
```

Program realizuje algorytm sortowania bąbelkowego. Dane do programu dla wygody wprowadzania najpierw są wprowadzane do Code Memory, po czym zostają przepisane do Xdata Memory. Pobierane są dwie kolejne liczby do rejestrów, a następnie są porównywane. Z powodu braku operacji porównywania, użyto odejmowania – jeśli wynik odejmowania jest ujemny to flaga C zostanie ustawiona na 1 i dokonuje się skok do fswap (zamiana miejscami elementów). Algorytm został zoptymalizowany przez ograniczenie iteracji pętli wewnętrznej o aktualny iterator pętli zewnętrznej.