Autorzy:

Maciej Ilów

Kinga Foksińska

Podstawy techniki mikroprocesowej

Sprawozdanie nr 4 i 5

Obsługa wyświetlacza LCD i zegar

czasu rzeczywistego z przerwaniami

Prowadzący – mgr inż. Antoni Sterna

Grupa K02-91m, środa TN, godz. 9:15

1.Cel ćwiczenia

Celem pierwszego ćwiczenia była nauka podstawowej obsługi wyświetlacza LCD (zestaw ZD537).

Wykonanie zadań wymagało również solidnego zapoznania się z dokumentacją zestawu ZD537 oraz zrozumienia sposobu komunikacji z wyświetlaczem.

Drugie ćwiczenie polegało na stworzeniu kodu realizującego zegar czasu rzeczywistego, który będzie realizowany w tle z użyciem przerwań.

2.Przebieg ćwiczenia

Laboratorium nr 4 składało się z 6 zadań o różnym poziomie trudności. Laboratorium nr 5 to procedury odpowiadające za konkretne działania w zegarze.

2.1 Zapis komendy

Funkcja "lcd_write_cmd" rozpoczyna się od przeniesienia zawartości rejestru akumulatora (ACC) na stos, aby zapisać wartość kodu komendy przed kolejnymi operacjami. Następnie, w rejestrze DPTR zapisywana jest informacja o statusie wyświetlacza. Kod tworzy pętlę, która sprawdza busy flag (BF), czyli sygnał oznaczający zajętość wyświetlacza. Pętla jest wykonywana dopóki BF nie zwróci wartości 0, co oznacza, że wyświetlacz nie jest zajęty. Gdy BF jest równy 0, w rejestrze DPTR przekazywana jest informacja o zapisie rejestru komend. Następnie, kod zabiera ze stosu wcześniej odłożony kod komendy i umieszcza go w rejestrowym wyświetlaczu. Na końcu, kod opuszcza funkcję "lcd_write_cmd" za pomocą instrukcji 'ret' (return).

```
;-----------------------------------------------------------------------
; Zapis komendy
;
; Wejscie: A - kod komendy
;-----------------------------------------------------------------------
lcd_write_cmd:
    push    ACC                 ; przeniesienie kodu komendy na stos, aby nie stracic go w dalszych krokach
    mov     DPTR, #RD_STAT      ; w rejestr DPTR przekazujemy informacje o statusie wyswietlacza
loop:                           ; petla odpowiada za sprawdzanie busy flag (BF), tak dlugo az nie zwróci 0 (wyswietlacz nie bedzie zajety)
    movx    A, @DPTR
    jb      ACC.7, loop

    mov     DPTR, #WR_CMD       ; w rejestr DPTR przekazujemy komorke z zapisem rejestru komend
    pop     ACC                 ; zabieramy ze stosu wczesniej odlozony kod komendy
    movx    @DPTR, A
    ret
```

2.2 . Zapis danych

Funkcja "lcd_write_data" służy do zapisywania danych na wyświetlaczu LCD. Funkcja działa analogicznie do „lcd_write_cmd”. Na początku, kod przenosi dane do zapisu z rejestru akumulatora (ACC) na stos, aby zachować je przed kolejnymi operacjami. W rejestrze DPTR zapisywana jest informacja o statusie wyświetlacza. Następnie, tworzona jest pętla, która sprawdza busy flag (BF), czyli sygnał oznaczający zajętość wyświetlacza. Pętla jest wykonywana dopóki BF nie zwróci wartości 0, co oznacza, że wyświetlacz nie jest zajęty. Gdy BF jest równy 0, w rejestrze DPTR przekazywana jest informacja o zapisie rejestru danych. Kod następnie pobiera wcześniej odłożone dane do zapisu ze stosu i umieszcza je w rejestrowym wyświetlaczu. Na końcu, kod opuszcza funkcję "lcd_write_data" za pomocą instrukcji 'ret' (return).

```
;-----------------------------------------------------------------------
; Zapis danych
;
; Wejscie: A - dane do zapisu
;-----------------------------------------------------------------------
lcd_write_data:
    push    ACC                 ; przeniesienie danych do zapisu na stos, aby nie stracic ich w dalszych krokach
    mov     DPTR, #RD_STAT      ; w rejestr DPTR przekazujemy informacje o statusie wyswietlacza
loop1:                          ; petla odpowiada za sprawdzanie busy flag (BF), tak dlugo az nie zwróci 0 (wyswietlacz nie bedzie zajety)
    movx    A, @DPTR
    jb      ACC.7, loop1

    mov     DPTR, #WR_DATA      ; w rejestr DPTR przekazujemy komorke z zapisem rejestru danych
    pop     ACC                 ; zabieramy ze stosu wczesniej odlozone dane do zapisu
    movx    @DPTR, A
    ret
```

2.3 Inicjowanie wyświetlacza

Funkcja "lcd_init" służy do inicjowania wyświetlacza LCD. Na początku, kod przekazuje informacje o ustawieniu dwóch linii wyświetlacza (N=1) do rejestru akumulatora (ACC), a następnie wywołuje funkcję "lcd_write_cmd", która zapisuje te informacje jako komendę. Kolejnym krokiem jest przekazanie komendy czyszczenia ekranu do akumulatora, po czym ponownie wywoływana jest funkcja "lcd_write_cmd". Wreszcie, kod przekazuje do akumulatora informacje o włączeniu wyświetlacza (LCD_ON) i ponownie wywołuje funkcję "lcd_write_cmd". Funkcja "lcd_init" kończy się instrukcją 'ret' (return), która opuszcza funkcję.

```
;-----------------------------------------------------------------------
; Inicjowanie wyswietlacza
;-----------------------------------------------------------------------
lcd_init:
    mov     A, #INIT_DISPLAY    ; przekazanie informacji o N=1, czyli ustawieniu dwoch linii wyswietlacza
    lcall   lcd_write_cmd

    mov     A, #CLEAR_DISPLAY   ; czyszczenie ekranu
    lcall   lcd_write_cmd

    mov     A, #LCD_ON          ; wlaczenie wyswietlacza
    lcall   lcd_write_cmd
    ret
```

2.4 Ustawienie bieżącej pozycji wyświetlania

Funkcja "lcd_gotoxy" służy do ustawienia bieżącej pozycji wyświetlania na wyświetlaczu LCD. Na wejściu przyjmuje wartość A, która reprezentuje pozycję na wyświetlaczu w formacie ---y | xxxx, gdzie y to numer wiersza, a xxxx to numer kolumny. Kod rozpoczyna się od przeniesienia wartości pozycji na wyświetlaczu z akumulatora do rejestru R7. W pierwszej części kodu skupiamy się na operacjach na starszych bitach (nie tracąc informacji o młodszych, dzięki odłożeniu pozycji wyświetlacza do rejestru R7). Za pomocą operacji AND (wyzerowania młodszych bitów) oraz dwukrotnego przesunięcia bitów w lewo (przesunięcie bitu czwartego na pozycję szóstą), kod ustawia w akumulatorze odpowiednią informację o wierszu. Operacja OR ustawia wartość 1000 0000 lub 1100 0000 w akumulatorze, w zależności od wiersza. Ta wartość zostaje zapisana w rejestrze R6.

Kod przywraca wartość początkowej pozycji na wyświetlaczu do akumulatora, a następnie wykonuje operację AND, aby wyzerować cztery starsze bity. Potem, za pomocą operacji OR, łączy cztery młodsze bity z wartością wcześniej zmienionych czterech starszych bitów. Na końcu wywołuje funkcję "lcd_write_cmd" z ostateczną reprezentacją pozycji w akumulatorze, a następnie opuszcza funkcję "lcd_gotoxy" za pomocą instrukcji 'ret' (return).

```
;------------------------------------------------------------------------
; Ustawienie biezacej pozycji wyswietlania
;
; Wejscie: A - pozycja na wyswietlaczu: ---y | xxxx
;------------------------------------------------------------------------
lcd_gotoxy:
;   anl     A, #00011111b
;   jnb     ACC.4, wyslij
;   clr     ACC.4
;   add     A, #40h
; wyslij:
;   orl     A, #80h             ; operacja sprawi, ze w akumulatorze otrzymamy 1000 0000 lub 1100 0000
;   lcall   lcd_write_cmd


    mov     R7, A               ; odlozenie pozycji na wyswietlaczu do rejestru R7
    anl     A, #10h             ; operacja sprawi, ze w akumulatorze otrzymamy 0000 0000 lub 0001 0000 zaleznie od poczatkowego y
    rl      A                   ; podwojne przesuniecie bitow w lewo, dzieki czemu wynikiem operacji bedzie 0000 0000 lub 0100 0000
    rl      A
    orl     A, #80h             ; operacja sprawi, ze w akumulatorze otrzymamy 1000 0000 lub 1100 0000
                                ; dzieki powyzszym operacjom mamy ustawiona docelowa informacja na temat wiersza do wyswietlenia
    mov     R6, A
    mov     A, R7               ; ponowne wziecie poczatkowej pozycji na wyswietlaczu do akumulatora
    anl     A, #0Fh             ; wyzerowanie czterech starszych bitow
    orl     A, R6               ; stworzenie reprezentacji skladajacej sie z wczesniej zmienionych 4 starszych bitow oraz niezmienionych 4 mlodszych bitow
    lcall   lcd_write_cmd
    ret
```

2.5 Wyświetlenie tekstu od bieżącej pozycji

Funkcja "lcd_puts" służy do wyświetlania tekstu na wyświetlaczu LCD, zaczynając od bieżącej pozycji. Na wejściu przyjmuje wartość DPTR, która wskazuje adres pierwszego znaku tekstu w pamięci kodu.
Kod rozpoczyna się od wyzerowania akumulatora, a następnie pobiera znak tekstu, na który wskazuje DPTR. Sprawdza, czy znak jest zerem (co oznacza koniec napisu) – jeśli tak, przechodzi do etykiety "koniec" i opuszcza funkcję za pomocą instrukcji 'ret' (return).

Jeśli znak nie jest zerem, kod zapisuje wartości rejestru DPL i DPH na stosie, a następnie wywołuje funkcję "lcd_write_data", która wysyła dane o znaku do sterownika. Po wykonaniu tej funkcji, wartości DPL i DPH są pobierane ze stosu. Następnie, DPTR jest inkrementowany, aby wskazać kolejny znak napisu. Kod wykonuje skok do początku funkcji "lcd_puts" i powtarza te kroki, dopóki nie napotka znaku zerowego.

```
;------------------------------------------------------------------------
; Wyswietlenie tekstu od biezacej pozycji
;
; Wejscie: DPTR - adres pierwszego znaku tekstu w pamieci kodu
;------------------------------------------------------------------------
lcd_puts:
    clr     A
    movc    A, @A+DPTR          ; pobranie znaku tekstu na ktory wskazuje DPTR
    jz      koniec              ; sprawdzenie czy jest zerem (czy napis nie jest zakonczony)
    push    DPL
    push    DPH
    lcall   lcd_write_data      ; wyslanie danych o znaku do sterownika
    pop     DPH
    pop     DPL
    inc     DPTR                ; przejscie do kolejnego znaku napisu
    sjmp    lcd_puts
koniec:
    ret
```

2.6 Wyświetlenie liczby dziesiętnej

Funkcja "lcd_dec_2" służy do wyświetlania liczby dziesiętnej na wyświetlaczu LCD. Na wejściu przyjmuje wartość A, która reprezentuje liczbę do wyświetlenia (zakres od 00 do 99). Kod rozpoczyna się od wstawienia dzielnika (liczby 10) do rejestru B, a następnie dzieli zawartość akumulatora przez zawartość rejestru B za pomocą instrukcji 'div AB'.

Po dzieleniu, w akumulatorze znajduje się wynik dzielenia, do którego dodawany jest znak '0' (w reprezentacji ASCII), przekształcając go na kod ASCII. Następnie wywołuje funkcję "lcd_write_data", aby wyświetlić cyfrę dziesiątek liczby.

Kod przenosi resztę z dzielenia do akumulatora, a następnie dodaje do niej znak '0' (w reprezentacji ASCII), konwertując resztę na kod ASCII. Wywołuje funkcję "lcd_write_data" ponownie, aby wyświetlić cyfrę jedności liczby. Na końcu opuszcza funkcję "lcd_dec_2" za pomocą instrukcji 'ret' (return).

```
;------------------------------------------------------------------------
; Wyswietlenie liczby dziesietnej
;
; Wejscie: A - liczba do wyswietlenia (00 ... 99)
;------------------------------------------------------------------------
lcd_dec_2:
    mov     B, #10              ; wstawienie do rejestru B dzielnika (liczby 10)
    div     AB                  ; dzielenie zawartosci akumulatora przez zawartosc rejestru B
    add     A, #'0'             ; w akumulatorze znajduje sie wynik dzielenia, dodajemy do niego '0' dzieki czemu przeksztalcamy go na kod ascii
    lcall   lcd_write_data

    mov     A, B                ; przenosimy reszte z dzielenia do akumulatora
    add     A, #'0'             ; konwertujemy reszte z dzielenia na kod ascii
    lcall   lcd_write_data
    ret
```

2.7 Zegar czasu rzeczywistego – inicjowanie Timera 0 w trybie 16-bitowym z przerwaniami

Ten fragment kodu odpowiada za inicjalizację Timera 0 w trybie 16-bitowym z obsługą przerwań. Procedura timer_init wykonuje następujące kroki: Na początku kod zatrzymuje Timer 0, ustawiając TR0 na 0, aby zapobiec jego działaniu. Następnie, instrukcja anl TMOD, #11110000b wykonuje operację logicznego AND na rejestrze TMOD, aby wyzerować bity od 0 do 3, które odpowiadają za tryb Timera 0. Instrukcja orl TMOD, #00000001b ustawia tryb Timera 0 jako 16-bitowy, ustawiając wartość 1 w najmłodszym bicie (bit 0) TMOD. Kolejnym krokiem jest wpisanie wartości do rejestrów licznika Timer 0, TL0 i TH0, które są zdefiniowane jako stała LOAD. Wartość ta określa liczbę cykli licznika, po której nastąpi przerwanie. Po tym, kod wyzerowuje flagę przepełnienia timera (TF0), która informuje o przekroczeniu maksymalnej wartości licznika. Następnie, ustawiana jest flaga ET0, odblokowująca przerwania Timera 0, oraz flaga EA, globalnie odblokowująca przerwania w programie. W końcowym kroku, kod ustawia TR0 na 1, uruchamiając Timer 0 i rozpoczynając jego działanie. Na koniec procedura zwraca sterowanie do miejsca, z którego została wywołana.

```
;-----------------------------------------------------------------------
; Inicjowanie Timera 0 w trybie 16-bitowym z przerwaniami
;-----------------------------------------------------------------------
timer_init:
    clr     TR0                         ; zatrzymanie timera
    anl     TMOD, #11110000b            ; wyzerowanie timera 0
    orl     TMOD, #00000001b            ; ustawienie tmod w tryb 1 (16-bitowy)
    mov     TL0, #LOW(LOAD)             ; wpisanie wartosci do rejestrow licznika
    mov     TH0, #HIGH(LOAD)
    clr     TF0                         ; wyzerowanie flagi przepelnienia timera
    setb    ET0                         ; odblokowanie przerwan Timera 0
    setb    EA                          ; globalne odblokowanie przerwan
    setb    TR0                         ; uruchomienie timera
    ret
```

2.8 Zegar czasu rzeczywistego – obsługa przerwania Timera 0

Kod obsługi przerwania Timera 0 wykonuje się w odpowiedzi na przerwanie wywołane przez ten timer. Głównym celem tego kodu jest zliczanie czasu i aktualizacja wartości godzin, minut i sekund. W pierwszym kroku, kod inicjalizuje rejestr licznika Timer 0 i ustawia odpowiednie wartości dla zliczania czasu. Następnie, w obszarze przerwania, kod inkrementuje zmienną CNT_100, która reprezentuje setne części sekundy. Jeśli wartość CNT_100 osiągnie wartość graniczną 100, zostaje wyzerowana, a flaga SEC_CHANGE jest ustawiana, co sygnalizuje zmianę sekundy. Kod następnie sprawdza wartości sekund, minut i godzin. Jeśli osiągnięta zostanie wartość graniczna (60 dla sekund, minut i 24 dla godzin), odpowiednie zmienne są zerowane, co oznacza przechodzenie na kolejną jednostkę czasu. Na koniec obsługi przerwania, kod przywraca poprzedni stan rejestrów i powraca do głównego programu.

```
;-----------------------------------------------------------------------
; Obsluga przerwania Timera 0
;-----------------------------------------------------------------------
T0_int:
    mov     TL0, #LOW(LOAD)             ; wpisanie wartosci do rejestrow licznika
    mov     TH0, #HIGH(LOAD)
    push    ACC
    push    PSW

    clr     SEC_CHANGE                  ; wyzerowanie flagi zmiany sekund
    inc     CNT_100
    mov     A, CNT_100
    cjne    A, #100, koniec             ; sprawdzenie, czy osiagnal wartosc graniczna
    mov     CNT_100, #0                 ; wyzerowanie, jesli tak sie stalo
    setb    SEC_CHANGE                  ; ustawienie flagi zmiany sekund
    inc     SEC
    mov     A, SEC
    cjne    A, #60, koniec
    mov     SEC, #0
    inc     MIN
    mov     A, MIN
    cjne    A, #60, koniec
    mov     MIN, #0
    inc     HOUR
    mov     A, HOUR
    cjne    A, #24, koniec
    mov     HOUR, #0

koniec:
    pop     PSW
    pop     ACC
    reti
```

2.9 Zegar czasu rzeczywistego – inicjowanie zmiennych związanych z czasem

Ten fragment kodu odpowiada za inicjalizację zmiennych związanych z czasem. Procedura clock_init wykonuje następujące kroki: Najpierw kod przypisuje wartość 0 do zmiennej CNT_100, która reprezentuje setne części sekundy. Jest to zmienna używana do śledzenia upływu czasu z dokładnością do setnych części sekundy. Następnie, kod przypisuje wartości graniczne do zmiennych SEC, MIN i HOUR. W przypadku sekund, wartość 59 reprezentuje maksymalną wartość, przed przejściem na kolejną minutę. W przypadku minut, wartość 59 reprezentuje maksymalną wartość przed przejściem na kolejną godzinę. Natomiast w przypadku godzin, wartość 23 reprezentuje maksymalną wartość przed przejściem na kolejny dzień.

Dodatkowo, kod inicjalizuje zmienne ALARM_SEC, ALARM_MIN i ALARM_HOUR, które są używane do ustawienia alarmu. W tym przypadku, wartość 0 dla ALARM_SEC i ALARM_MIN, oraz wartość 6 dla ALARM_HOUR, oznaczają, że alarm jest ustawiony na godzinę 6:00.

Na koniec procedura zwraca sterowanie do miejsca, z którego została wywołana.

```
;------------------------------------------------------------------------
; Inicjowanie zmiennych zwiazanych z czasem
;------------------------------------------------------------------------
clock_init:
    mov     CNT_100, #0
    mov     SEC, #59
    mov     MIN, #59
    mov     HOUR, #23

    mov     ALARM_SEC, #0
    mov     ALARM_MIN, #0
    mov     ALARM_HOUR, #6
    ret
```

2.10 Zegar czasu rzeczywistego – wyświetlanie czasu

Ten fragment kodu wykonuje procedurę clock_display, która służy do wyświetlania aktualnego czasu na wyświetlaczu LCD. Na początku, kod wykonuje instrukcję mov A, #CLEAR_DISPLAY, która ustawia wartość CLEAR_DISPLAY w rejestrze akumulatora A. Ta wartość reprezentuje komendę czyszczenia ekranu na wyświetlaczu. Następnie, kod wywołuje procedurę lcd_write_cmd, która zapisuje wartość z rejestru akumulatora A jako komendę do wyświetlacza, co powoduje wyczyszczenie ekranu. Po wyczyszczeniu ekranu, kod przechodzi do pozycji (4, 0) na wyświetlaczu, wywołując procedurę lcd_gotoxy z odpowiednimi parametrami. Kolejne kroki dotyczą wyświetlania godzin, minut i sekund na wyświetlaczu. W przypadku godzin, wartość HOUR jest przenoszona do rejestru akumulatora A i następnie wywoływana jest procedura lcd_dec_2, która konwertuje wartość na postać dwucyfrową i wyświetla ją na wyświetlaczu. Następnie, kod sprawdza parzystość sekund, poprzez sprawdzenie najmłodszego bitu rejestru akumulatora A (bit ACC.0). W zależności od parzystości, kod ustawia odpowiednią wartość (dwukropek lub spacje) w rejestrze akumulatora A i wywołuje procedurę lcd_write_data, która zapisuje wartość do wyświetlacza. Analogiczne kroki są powtarzane dla minut i sekund. Na koniec procedura zwraca sterowanie do miejsca, z którego została wywołana.

```
;-----------------------------------------------------------------------
; Wyswietlanie czasu
;-----------------------------------------------------------------------
clock_display:
    mov     A, #CLEAR_DISPLAY           ; czyszczenie ekranu
    lcall   lcd_write_cmd

    mov     A, #04h                     ; x = 4, y = 0
    lcall   lcd_gotoxy                  ; przejscie do pozycji (4, 0)

    mov     A, HOUR                     ; wyswietlenie godzin
    lcall   lcd_dec_2

    mov     A, SEC
    jb      ACC.0, dwukropek1           ; sprawdzenie parzystosci sekund
    mov     A, #' '                     ; jesli parzysta wyswietl spacje
    sjmp    write_sign1
dwukropek1:
    mov     A, #':'                     ; jesli nieparzysta wyswietl dwukropek
write_sign1:
    lcall   lcd_write_data

    mov     A, MIN                      ; wyswietlanie minut
    lcall   lcd_dec_2

    mov     A, SEC
    jb      ACC.0, dwukropek2           ; sprawdzenie parzystosci sekund
    mov     A, #' '                     ; jesli parzysta wyswietl spacje
    sjmp    write_sign2
dwukropek2:
    mov     A, #':'                     ; jesli nieparzysta wyswietl dwukropek
write_sign2:
    lcall   lcd_write_data

    mov     A, SEC                      ; wyswietlanie sekund
    lcall   lcd_dec_2

    ret
```

2.11 Zegar czasu rzeczywistego – obsługa alarmu

Ten fragment kodu wykonuje procedurę clock_alarm, która jest odpowiedzialna za obsługę alarmu w programie. Na początku, kod porównuje wartość HOUR (godziny), MIN (minuty) i SEC (sekundy) z odpowiednimi zmiennymi ALARM_HOUR, ALARM_MIN i ALARM_SEC, które określają ustawiony czas alarmu. Jeśli wartości godziny, minut i sekund są zgodne z ustawionym alarmem, kod kontynuuje wykonanie. W przeciwnym razie, przechodzi do etykiety clear_alarm, która jest odpowiedzialna za wyłączenie alarmu. W przypadku, gdy alarm zostaje rozpoznany, kod wyłącza diodę LED (bit 7 w rejestrze LEDS) poprzez ustawienie wartości CLR. Następnie, kod przechodzi do pozycji (5, 1) na wyświetlaczu LCD, wywołując procedurę lcd_gotoxy. Kolejnym krokiem jest wyświetlenie tekstu zdefiniowanego w etykiecie text_alarm na wyświetlaczu, wykorzystując procedurę lcd_puts. Po zakończeniu obsługi alarmu, kod przechodzi do etykiety end_alarm, a następnie zwraca sterowanie do miejsca, z którego został wywołany.

```
;-----------------------------------------------------------------
; Obsluga alarmu
;-----------------------------------------------------------------
clock_alarm:
    mov     A, HOUR
    cjne    A, ALARM_HOUR, clear_alarm  ; sprawdzenie godziny
    mov     A, MIN
    cjne    A, ALARM_MIN, clear_alarm   ; sprawdzanie minut
    mov     A, SEC
    cjne    A, ALARM_SEC, clear_alarm   ; sprawdzenie sekund

    clr     LEDS.7                      ; ustawienie diody led jesli jest alarm
    mov     A, #15h                     ; x = 5, y = 1
    lcall   lcd_gotoxy                  ; przejscie do pozycji (5, 1)
    mov     DPTR, #text_alarm           ; wyswietlenie tekstu
    lcall   lcd_puts

    sjmp    end_alarm
clear_alarm:
    setb    LEDS.7                      ; wygaszenie diody led jesli nie ma alarmu

end_alarm:
    ret
```

2.12 Zegar czasu rzeczywistego - pętla główna programu

Ten fragment kodu reprezentuje pętlę główną programu. Wykonuje się ona w niekończącej się pętli, gdzie głównym zadaniem jest monitorowanie zmiennej SEC_CHANGE i wykonywanie odpowiednich działań w zależności od jej wartości. Na początku pętli, instrukcja jnb sprawdza, czy zmienna SEC_CHANGE jest równa 0. Jeśli tak, oznacza to brak zmiany sekundy i program pozostaje w pętli, oczekując na zmianę. Jeśli wartość SEC_CHANGE jest różna od 0, kod wyzerowuje zmienną SEC_CHANGE, aby przygotować ją na kolejną zmianę sekundy. Następnie, lcall clock_display wywołuje procedurę odpowiedzialną za wyświetlanie aktualnego czasu. Po wyświetleniu czasu, lcall clock_alarm wywołuje procedurę obsługi alarmu, która może zawierać odpowiednie działania, gdy wystąpi alarm w programie. Na końcu pętli, instrukcja sjmp main_loop powoduje powrót do początku pętli głównej, rozpoczynając kolejny cykl.

```
;-----------------------------------------------------------------
; Petla glowna programu
;-----------------------------------------------------------------
main_loop:
    jnb     SEC_CHANGE, main_loop       ; czekanie jesli SEC_CHANGE = 0
    clr     SEC_CHANGE                  ; wyzerowanie SEC_CHANGE
    lcall   clock_display               ; wyswietlenie czasu
    lcall   clock_alarm                 ; obsluga alarmu
    sjmp    main_loop
```

3.Wnioski

Podczas ćwiczeń udało nam się wykonać wszystkie obowiązkowe zadania z listy.

Największym problemem okazała się dla nas funkcja wyświetlająca tekst od bieżącej pozycji ze względu na to, że początkowo nie zadbaliśmy o ochronę wartości rejestru DPTR, który został użyty w funkcji lcd_write_data. A zatem po pobraniu pierwszego znaku napisu zniszczona została informacja o kolejnych znakach.

Dokładne zapoznanie się z dokumentacją zestawu ZD537 umożliwiło nam zrozumienie zasad działania wyświetlacza LCD. W szczególności skupiliśmy się na początkowych ustawieniach wyświetlacza oraz sposobie, aby je zmienić (np. ustawienie dwóch linii). Niezbędna okazała się przy tym tabela 6 z dokumentacji przedstawiająca instrukcje.

Dzięki zadaniom z tej listy zgłębiliśmy się również w temat poprawnego podłączenia wyświetlacza oraz znaczenia portów E, R/W i RS.

Przy wykonywaniu laboratorium polegającego na tworzeniu zegara rzeczywistego niezbędne okazało się dokładne zaznajomienie i rozumienie zadań z poprzedniej listy, które ułatwiły nam w znacznym stopniu pracę nad kolejnymi zadaniami.