Projekt z Architektury Systemów Komputerowych

Wykonawca: Natalia Góras

Tytuł pracy: Symulator działania sygnalizacji świetlnej z sekundnikiem. Do tego celu należy wykorzystać urządzenia peryferyjne świateł i wyświetlacze siedmiosegmentowe. Program musi korzystać z zegara czasu rzeczywistego.

Przygotowane:

Praca została wykonana w symulatorze procesora 8086. Aplikacja, która daje nam możliwość przeprowadzenia badania to sms32v50. Przed przystąpieniem do ćwiczenia musimy zainstalować program oraz przystosować go do naszego systemu operacyjnego. W moim przypadku jest to Windows, więc instalacja nie wymaga dodatkowych sterowników. Gdy nasz program działa przechodzimy do dalszej części.

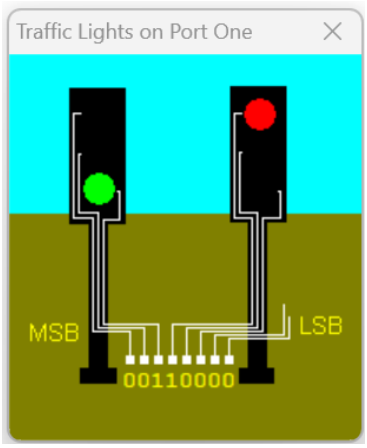
Założenia do pracy:

Moje zadanie polega na stworzeniu sygnalizacji świetlnej oraz podłączeniu do niej zegara z czasem rzeczywistym. Sygnalizacja świetlna będzie przedstawiała na wyświetlaczu dwa slupy sygnalizacyjne.
Po lewej stronie sygnalizacja, będzie odpowiadała za drogę podporządkowaną.
Po prawej stronie sygnalizacja reprezentuje drogę główną. Do drogi głównej będziemy mieli podłączony zegar, który będzie odmierzał czas do zmiany koloru światła.
Pamiętać trzeba jednak, że w rzeczywistości świata zielone nie zmieniają się w takich samych odstępach czasu dla każdej z dróg. U nas droga główna będzie miała o 3s dłużej światło zielone niż droga podporządkowana. Wyświetlacz siedmiosegmentowy będzie zmniejszał czas do zmiany koloru światła.

Droga podporządkowana	Droga główna
Światło czerwone: 9s	Światło czerwone: 6s
Światło pomarańczowoczerwone:2s	Światło pomarańczowoczerwone:2s
Światło zielone: 6s	Światło zielone: 9s
Światło pomarańczowe: 2s	Światło pomarańczowe: 2s

Sygnalizacja świetlna krótki wstęp:

Sama sygnalizacja świetlna, wykorzystuje symulator świateł drogowych, który podłączony jest do portu nr 01. Gdy program wysyła nam bity, automatycznie otwiera nam się okno odpowiedzialne za dany symulator. Prezentuje ono z założenia dwa slupy sygnalizacji, które w zadaniu będą reprezentować dwie różne drogi. W naszym przypadku jest to okno „Traffic Lights", przedstawione poniżej.



Rysunek 1 „Traffic Lights"

Dzięki portu o numerze 01 nasz program jest w stanie wysyłać bajty, które są odpowiedzialne za uruchomienie naszej sygnalizacji. Każdy bajt jest odpowiedzialny za konkretny kolor światła, więc wymaga to dużo precyzji przy obliczeniach oraz zamianie systemu dwójkowego na szesnastkowy, który jest wymagany w programie.

Jak zapisać bajty by uruchomić światło zielone oraz czerwone w jednym momencie?

Rysunku nr1. przedstawia nam dwie sygnalizacje, które mają światło zielone (droga podporządkowana) oraz światło czerwone (droga z pierwszeństwem). Jak widać od każdego wyświetlacza prowadzi przewód, który jest przypisany odpowiedniemu miejscu na bajcie. Bajt składa się z 8 bitów, pierwsze 3 odpowiedzialne są za lewą sygnalizację, a kolejne 3 za prawą. W takim wypadku zostają nam jeszcze 2 miejsca (najstarsze), które nie są używane. Sam zapis polega na wpisaniu odpowiednio „1" w miejscu do którego prowadzi przewód od danego wyświetlacza.

Jak widać po lewej stronie świeci się zielone światło, jego miejsce jest przypisane na 3 bicie. A po prawej świeci się światło czerwone, które jest najwyżej położone w sygnalizacji, a więc prowadzi do pierwszego miejsca należącego w bajce do drugiego wyświetlacza (czyli 4 bit).
Takie ustawienie świateł w systemie dwójkowym zapisane jest : 00110000. Przeliczając go system szesnastkowy równy jest 30. By wyświetlić światło w programie wystarczy wpisać:

```
MOV AL,30
OUT 01
```

Pierwsza linia odpowiedzialna jest za wpisanie w systemie szesnastkowym świateł, których oczekujemy na sygnalizacji, a druga odpowiada za wyświetlenie ich na odpowiednim porcie.

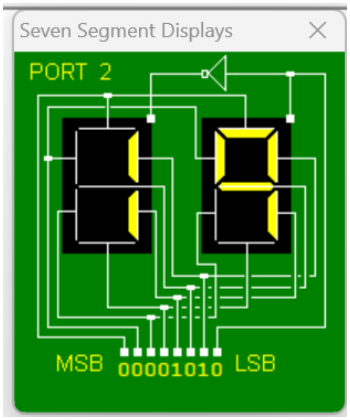
Był to krótki wstęp i wyjaśnienie na czym polega ogólnie sygnalizacja świetlna i jak będzie przeze mnie zapisywana w programie.

Tutaj przedstawię cały kod jaki został zastosowany do sygnalizacji świetlnej dla dwóch dróg, który będzie rozwinięty w dalszej części zadania. Oczywiście nie wspomniałam o tym wcześniej, lecz jak widać niżej, formuła zapisana jest w pętli, ponieważ światła maja za zadanie zmieniać się cały czas do momentu zatrzymania programu,

```
Start:
MOV AL,84 ; 1000 0100
OUT 01
MOV AL,C8 ; 1100 1000
OUT 01
MOV AL,30 ; 0011 0000
OUT 01
MOV AL,58 ; 0101 1000
OUT 01
JMP Start
END
```

Wyświetlacz siedmiosegmentowy krótki wstęp:

Jest to nasz kolejny wyświetlacz z jakim będziemy mieli odczynienia w zadaniu. Jest on podłączony do portu nr 02. Po uruchomieniu takiego programu pojawi się okno „Seven Segment Displays".



Rysunek 2 "Seven Segment Displays"

Program także działa na bajtach, które trzeba przekonwertować z systemu dwójkowego na system szesnastkowy. Sam schemat jest bardzo podobny do świateł, lecz tutaj każde miejsce bajtu odpowiada za kreskę (świecąca podłużna dioda), która zapali się w

momencie gdy jej przewód poprowadzony na odpowiadające jemu miejsce będzie miał wpisaną „1”. Po wysłaniu bajtu do portu, tam gdzie znajduje się jedynka, odpowiedni segment zapali się.
Trzeba pamiętać, że pierwszy bit z prawej strony określa, która z dwóch grup będzie aktywna. Jest to prosty przykład multipleksowania. Jeżeli najmniej znaczący bit wynosi „0”, to segmenty po lewej stronie będą aktywne. Jeżeli bit najmniej znaczący ma wartość „1”, to będą aktywne segmenty po prawej stronie.

Wyjaśnienie na przykładzie:

Chcemy by zapaliła się liczba jeden w lewym segmencie. Pierwszy krok to zauważenie, gdzie nitka prowadząca od „diody” ma swoje miejsce na bajcie. Gdy znajdziemy to miejsce wpisujemy tam „1”. A następnie przechodzimy do określenia segmentu, który ma się zaświecić. W przykładzie chcemy mieć oświetlony lewy segment, więc ostatni bit będzie oznaczony „0”.
Nasz bajt w takim razie ma postać: 00001010, zamieniając go na system szesnastkowy mamy wynik „A”. Zapis takiej liczby oraz jej wyświetlenie w programie zapisujemy następująco:

```
MOV AL,A
OUT 02
```

Pierwsza linia odpowiedzialna jest za wpisanie w systemie szesnastkowym liczby, jakiej oczekujemy na segmencie, a druga odpowiada za wyświetlenie ich na odpowiednim porcie.

Analogicznie postępujemy z liczba na prawym segmencie, TYLKO pamiętamy o zmianie ostatniego bitu na „1”, który przekieruje nas na prawą stronę i zaświeci liczby po prawej stronie segmentu.

Nasz licznik będzie poruszał się pomiędzy 9-0 więc cala sygnalizacja świetlna będzie operowała na tych liczbach i będzie wyświetlana tylko na prawym segmencie. Zależnie od światła w danym momencie będzie zmniejszanie liczb, od równej liczby początkowej . Kod do liczb oraz wywoływanie ich przedstawiam poniżej:

```
LIST9:
MOV AL,CF
OUT 02
MOV AL,FF
OUT 02
MOV AL,CB
OUT 02

LIST6:
MOV AL,FD
OUT 02
MOV AL,DD
OUT 02
MOV AL,4F
OUT 02
MOV AL,9F
```

```
OUT 02

LIST2:
MOV AL,B7
OUT 02
MOV AL,B
OUT 02
MOV AL,1
OUT 02
```

Przerwania sprzętowe

Są to krótkie fragmenty kodu, dostarczające użyteczne funkcje, które mogą być wywołane przez elementy sprzętu. W odróżnieniu od przerwań programowych, przerwania sprzętowe są asynchroniczne i mogą wystąpić w środku wykonania instrukcji, co wymaga dodatkowej uwagi przy programowaniu. Numer przerwania to w rzeczywistości adres wektora przerwania. Na przykład żądanie przerwania 02 spowoduje odczytanie przez procesor wartości komórki o adresie 02. Następnie wskaźnik instrukcji jest odkładany na stosie, a program kontynuuje pracę od instrukcji znajdującej się pod adresem wskazanym przez odczytaną wartość. W ten sposób wykonywany jest kod przerwania. Jego wykonanie kończy instrukcja IRET, która powoduje powrót z przerwania
Takie zastosowanie zastosowałam również w swoim programie by ulepszyć i wyeliminować możliwe błędy.

Schemat działania programu:

1. *Inicjalizacja programu:*

   - Zdefiniowanie danych w pamięci: DB 84, DB C8, DB 30, DB 58.

2. *Przypisanie wartości do rejestru BL:*

   - Rejestr BL ustawiany na wartość 02.

3. *Pierwsza pętla (ZIEL):*

   - Załadowanie wartości z pamięci (adres wskazywany przez BL = 02) do rejestru AL. Wysłanie wartości z rejestru AL na port 01. Skok do etykiety LIST9.

4. *LIST9:*

   - Wysłanie kilku wartości na port 02:
     - MOV AL, CF → OUT 02
     - MOV AL, FF → OUT 02
     - MOV AL, CB → OUT 02.

5. *Druga pętla (ZOL1):*

- Rejestr BL ustawiany na 03. Załadowanie wartości z pamięci do rejestru AL. Wysłanie wartości z rejestru AL na port 01. Skok do etykiety LIST2.

6. *LIST2:*

   - Wysłanie kilku wartości na port 02:
     - MOV AL, B7 → OUT 02
     - MOV AL, B → OUT 02
     - MOV AL, 1 → OUT 02.

7. *Trzecia pętla (CZER):*

   - Rejestr BL ustawiany na 04. Załadowanie wartości z pamięci do rejestru AL. Wysłanie wartości z rejestru AL na port 01. Skok do etykiety LIST6.

8. *LIST6:*

   - Wysłanie kilku wartości na port 02:
     - MOV AL, FD → OUT 02
     - MOV AL, DD → OUT 02
     - MOV AL, 4F → OUT 02
     - MOV AL, 9F → OUT 02

9. *Czwarta pętla (ZOL2):*

   - Rejestr BL ustawiany na 05. Załadowanie wartości z pamięci do rejestru AL. Wysłanie wartości z rejestru AL na port 01. Skok do etykiety LIST2.

10. *Powrót do początku:*

    - Sprawdzenie wartości w rejestrze BL:
      - Jeśli BL == 02, skok do ZOL1.
      - Jeśli BL == 03, skok do CZER.
      - Jeśli BL == 04, skok do ZOL2.
      - Jeśli BL == 05, skok do START.

11. *Kończenie programu:*

    - Instrukcja IRET kończy działanie programu i zwraca kontrolę do systemu.

12. *Zakończenie programu:*

    - Instrukcja END kończy kod programu.

Kod programu

```
JMP Start

DB 84
DB C8
DB 30
DB 58

Start:
        MOV BL,02

ZIEL:
        MOV AL, [BL]
        OUT 01
        JMP LIST9
ZOL1:
        MOV BL,03
        MOV AL,[BL]
        OUT 01
        JMP LIST2
CZER:
        MOV BL,04
        MOV AL,[BL]
        OUT 01
        JMP LIST6
ZOL2:
        MOV BL,05
        MOV AL,[BL]
        OUT 01
        JMP LIST2

STI
LIST9:
MOV AL,CF
OUT 02
MOV AL,FF
OUT 02
MOV AL,CB
OUT 02

LIST6:
MOV AL,FD
OUT 02
MOV AL,DD
OUT 02
MOV AL,4F
OUT 02
MOV AL,9F
OUT 02

LIST2:
MOV AL,B7
OUT 02
MOV AL,B
OUT 02
MOV AL,1
OUT 02

CMP BL,02
JZ ZOL1
CMP BL,03
JZ CZER
CMP BL,04
JZ ZOL2
CMP BL,05
JZ START

IRET
END
```