**Dokumentacja Projektu**

**Tytuł**: Program odwracający kolejność wyrazów we wprowadzonym tekście

**Autor**: Adrianna Dziewulska

**Numer indeksu**: 91261

**Przedmiot**: Architektura Systemów Komputerowych

**Prowadzący**: Wojciech Nabiałek

**Data**: 13.12.2024

Spis treści:

[1. Cel projektu………………………………………………………………………………………………………………3](#_Toc184987825)

[2. Opis działania programu……………………………………………………………………………………………3](#_Toc184987826)

[3. Wykorzystane narzędzia i mechanizmy……………………………………………………………………...4](#_Toc184987827)

[4. Lista kroków działania programu……………………………………………………………..………………..4](#_Toc184987828)

[5. Kod programu……………………………………………………………………………….……...…………………..7](#_Toc184987829)

[6. Wyniki działania………………………………………………………………………….……………...…………….9](#_Toc184987830)

[7. Wnioski…………………………………………………………………………………………………………………….9](#_Toc184987831)

[8. Bibliografia……………………………………………………………………………………………………...………..9](#_Toc184987832)

1. Cel projektu

Celem projektu jest stworzenie programu w Symulatorze Procesora 8086, który:

* Odwraca kolejność wyrazów we wprowadzonym z klawiatury zdaniu, które kończy się kropką.
* Jeżeli wyrazy zostaną odwrócone, zapali się zielone światło na sygnalizatorze.
* Jeżeli wyrazy nie mogą zostać odwrócone z powodu braku separatorów, zapali się czerwone światło na sygnalizatorze.

Program ma na celu praktyczne zastosowanie możliwości środowiska Symulatora Procesora 8086 w analizie tekstu oraz obsłudze urządzeń zewnętrznych.

1. Opis działania programu

Program rezerwuje przestrzeń w pamięci na bufor, w którym będą przechowywane dane. Wielkość bufora odpowiada rozmiarowi wyświetlacza VDU. Tekst wprowadzany przez użytkownika za pomocą klawiatury jest odczytywany dzięki funkcjom symulatora, które przetwarzają wpisane znaki.

Program kontroluje wprowadzone znaki pod kątem obecności separatorów (spacji i kropek) oraz sprawdza, czy ich liczba mieści się w przydzielonym buforze. Każde zdanie musi kończyć się kropką. Każdy wprowadzony znak jest tymczasowo przechowywany w buforze pamięci. Wyrazy są rozpoznawane na podstawie spacji poprzedzającej ich początek lub,   
w przypadku pierwszego wyrazu, braku wcześniejszych znaków.

Całe wyrazy są następnie zapisywane na stos, co umożliwia ich odwrócenie podczas odczytu. Odwrócony tekst jest wyświetlany na wyświetlaczu VDU. W przypadku pomyślnego odwrócenia całego zdania, zapala się zielone światło na sygnalizatorze,   
a odwrócone zdanie zostaje wyświetlone na wyświetlaczu. Jeśli w tekście brakuje separatorów lub jego długość przekracza rozmiar bufora, na sygnalizatorze zapala się czerwone światło.

1. Wykorzystane narzędzia i mechanizmy
2. Symulator Procesora 8086: Środowisko programu.
3. Instrukcje wejścia/wyjścia: Do wczytania tekstu i jego wyświetlania.
4. Bufor pamięci: Do przechowywania danych wejściowych.
5. Stos: Do przechowywania wyrazów w trakcie odwracania kolejności.
6. Urządzenia zewnętrzne: Sygnalizator (zielone/czerwone światło).
7. Lista kroków działania programu
8. **Inicjalizacja bufora**:
   * Instrukcja MOV BL, 70 ustawia wskaźnik BL na początkowy adres bufora   
     w pamięci (70h).
9. **Start programu (pętla główna)**:
   * IN 00: Odczytuje znak wprowadzony przez użytkownika i zapisuje go do rejestru AL.
   * MOV [BL], AL: Zapisuje odczytany znak z AL do pamięci pod adresem wskazywanym przez BL.
   * INC BL: Zwiększa wskaźnik BL, przygotowując miejsce na kolejny znak w buforze.
10. **Sprawdzanie znaków kończących zdanie**:
    * CMP AL, 2E: Porównuje odczytany znak z kropką (.).
      + Jeśli znak to kropka (2E), przechodzi do etykiety Cout (odwracanie tekstu i wyświetlanie).
    * CMP AL, 20: Porównuje odczytany znak ze spacją ( ).
      + Jeśli znak to spacja (20), przechodzi do etykiety Inc (zwiększenie licznika wyrazów).
11. **Sprawdzanie końca bufora**:
    * CMP BL, B0: Sprawdza, czy wskaźnik BL osiągnął wartość B0 (koniec bufora).
      + Jeśli tak, przechodzi do etykiety Error (błąd bufora).
12. **Powrót do pętli głównej**:
    * JMP Start: Powraca do początku pętli, aby odczytać kolejny znak.
13. **Zwiększenie licznika wyrazów (Inc)**:
    * INC DL: Zwiększa licznik wyrazów (DL).
    * JMP Start: Powraca do początku pętli.
14. **Odwracanie tekstu (Cout)**:
    * CMP DL, 0: Sprawdza, czy licznik wyrazów wynosi zero.
      + Jeśli tak, przechodzi do etykiety Error (brak wyrazów).
    * DEC BL: Cofnięcie wskaźnika BL o jedno miejsce.
    * MOV DL, 0: Zeruje licznik wyrazów (DL).
    * MOV CL, C0: Ustawia wskaźnik CL na początek miejsca docelowego dla odwróconego tekstu (C0h).
15. **Pętla odczytu znaków (CoutIn)**:
    * MOV AL, [BL]: Pobiera znak z bufora wskazywanego przez BL do AL.
    * PUSH AL: Umieszcza znak na stosie.
    * INC DL: Zwiększa licznik znaków.
    * DEC BL: Przesuwa wskaźnik BL wstecz.
    * CMP AL, 20: Sprawdza, czy znak to spacja.
      + Jeśli tak, przechodzi do etykiety CoutPop (przenoszenie wyrazu z bufora na wyświetlacz).
    * CMP AL, 00: Sprawdza, czy znak to NULL (koniec danych).
      + Jeśli tak, przechodzi do etykiety CoutPopE.
    * JMP CoutIn: Kontynuuje odczytywanie znaków.
16. **Przenoszenie znaków ze stosu na wyświetlacz (CoutPop)**:
    * POP AL: Pobiera znak ze stosu.
    * MOV [CL], AL: Zapisuje znak w pamięci wskazywanej przez CL.
    * INC CL: Przesuwa wskaźnik CL do następnej pozycji.
    * DEC DL: Zmniejsza licznik znaków.
    * CMP DL, 0: Sprawdza, czy licznik znaków wynosi zero.
      + Jeśli tak, powraca do CoutIn.
    * JMP CoutPop: Kontynuuje przenoszenie znaków.
17. **Przenoszenie ostatniego znaku ze stosu (CoutPopE)**:
    * POP AL: Pobiera znak ze stosu.
    * MOV [CL], AL: Zapisuje znak w pamięci wskazywanej przez CL.
    * INC CL: Przesuwa wskaźnik CL.
    * DEC DL: Zmniejsza licznik znaków.
    * CMP DL, 0: Sprawdza, czy licznik znaków wynosi zero.
      + Jeśli tak, przechodzi do Done (zakończenie programu).
    * JMP CoutPop: Kontynuuje przenoszenie znaków.
18. **Obsługa błędów (Error)**:
    * MOV AL, 90: Ustawia kod błędu.
    * OUT 01: Wyświetla sygnał błędu (czerwone światło).
    * JMP Endl: Przechodzi do końca programu
19. **Zakończenie programu (Done)**:
    * MOV AL, 24: Ustawia kod zakończenia.
    * OUT 01: Wyświetla sygnał sukcesu (zielone światło).
    * JMP Endl: Przechodzi do końca programu
20. **Koniec programu (Endl)**:
    * END: Zatrzymuje program.
21. Kod programu
22. MOV BL, 70
23. Start:
24. IN 00
25. MOV [BL], AL
26. INC BL
27. CMP AL, 2E
28. JZ Cout
29. CMP AL, 20
30. JZ Inc
31. CMP BL, B0
32. JZ Error
33. JMP Start
34. Inc:
35. INC DL
36. JMP Start
37. Cout:
38. CMP DL, 0
39. JZ Error
40. DEC BL
41. MOV DL, 0
42. MOV CL, C0
43. CoutIn:
44. MOV AL, [BL]
45. PUSH AL
46. INC DL
47. DEC BL
48. CMP AL, 20
49. JZ CoutPop
50. CMP AL, 00
51. JZ CoutPopE
52. JMP CoutIn
53. CoutPop:
54. POP AL
55. MOV [CL], AL
56. INC CL
57. DEC DL
58. CMP DL, 0
59. JZ CoutIn
60. JMP CoutPop
61. CoutPopE:
62. POP AL
63. MOV [CL], AL
64. INC CL
65. DEC DL
66. CMP DL, 0
67. JZ Done
68. JMP CoutPop
69. Error:
70. MOV AL, 90
71. OUT 01
72. JMP Endl
73. Done:
74. MOV AL, 24
75. OUT 01
76. JMP Endl
77. Endl:
78. END
79. Wyniki działania
80. Przykład poprawnego działania:

* Wejście: Programowanie w assemblerze jest ciekawe.
* Wyjście: ciekawe. Jest assemblerze Programowanie
* Sygnalizator: Zielone światło.

1. Przykład błędnego działania:

* Wejście: Test.
* Sygnalizator: Czerwone światło.

1. Wnioski

Projekt zrealizowany w środowisku Symulatora Procesora 8086 pozwolił na:

* Praktyczne zastosowanie funkcji wejścia/wyjścia oraz stosu.
* Zrozumienie mechanizmów obsługi tekstu w środowisku symulującym mikroprocesory.
* Efektywne wykorzystanie sygnalizatora jako urządzenia zewnętrznego.

Program spełnił założenia projektowe, umożliwiając obsługę zarówno poprawnych, jak   
i błędnych danych wejściowych. Dodatkowe wyzwanie stanowiło sterowanie sygnalizatorem, które zostało poprawnie zaimplementowane.

1. Bibliografia
   1. **Dokumentacja Symulatora procesora 8086 na stronie** GitHub - Microprocessor Simulator: https://github.com/dwhinham/Microprocessor-Simulator/tree/master Dostęp: 13.12.2024.
   2. Materiały dostępne na stronie uniwersytetu: https://strefa.ii.uws.edu.pl/   
      Dostęp: 13.12.2024.