Interpreter Pseudoassemblera

Igor Faliszewski

Spis Treści

1. Wstęp

Interpreter Pseudoassemblera jest programem służącym do analizy poprawności i interpretacji kodu w języku Pseudoassemblera, opartym na języku Assembler, na kod maszynowy, oraz jego symulacji.

Program, po podaniu do wejścia kodu Pseudoassemblera, analizuje jego poprawność, wyświetlając użytkownikowi dokładne informacje o rodzaju ewentualnych błędów i linijki, w której on wystąpił. Jeżeli kod jest poprawny, program przystępuje do jego interpretacji na kod maszynowy i zapisuje wynik w nowoutworzonym pliku. Następnie uruchamiana jest symulacja wykonania programu. Przy podaniu do wejścia kodu w języku maszynowym symulacja uruchamia się od razu. Wynik symulacji jest zapisywany w oddzielnym nowoutworzonym pliku.

Program pozwala na wizualizację symulacji i analizę wykonania programu krok po kroku, dalej nazywane trybem debugowania.

2. Specyfikacja projektu

Program został napisany w środowisku Microsoft Visual Studio Enterprise 2019.

Zewnętrzne biblioteki, z których korzysta program to:

PDCurses: https://pdcurses.org/

3. Założenia języka Pseudoassemblera oraz kodu maszynowego

Kod w języku Pseudoassemblera, który jest przyjmowany przez program jest zgodny z założeniami języka Pseudoassemblera, z poniższymi uwagami:

- Maksymalna długość linii kodu to 100 znaków.
- Komentarzami niewpływającymi na interpretację programu są:
  - Początkowe linie pliku rozpoczęte dwoma ukośnikami.
  - Ciąg znaków po poprawnej linii kodu Pseudoassemblera rozpoczęty dwoma ukośnikami.
- Linie między instrukcjami zawierające tylko komentarz nie są dozwolone.
- Wykorzystanie ujemnego offsetu jest dozwolone.

Kod w języku maszynowym, który jest przyjmowany przez program jest zgodny z następującymi założeniami, dalej nazywanymi założeniami języka maszynowego:

- Sekcja danych składa się z linii zawierających cztery ciągi dwuznakowe oddzielone spacjami. Jeden ciąg oznacza jeden bajt w systemie szesnastkowym, jedna linia oznacza jedną komórkę pamięci.
- Gdy bajt w komórce pamięci ma wartość niezdefiniowaną, jest zapisywany jako dwie tyldy „~~”.
- Sekcja danych jest oddzielona od sekcji rozkazów jedną pustą linią.
- Sekcja rozkazów składa się z linii zawierających cztery lub dwa ciągi dwuznakowe oddzielone spacjami. Jeden ciąg oznacza jeden bajt w systemie szesnastkowym. Jedna linia reprezentuje jedną instrukcję. Ilość ciągów w linii jest zależna od długości instrukcji, którą linia reprezentuje.
- Komentarze nie są dozwolone.

4. Wejście

Program jako poprawne wejście do analizy poprawności kodu Pseudoassemblera przyjmuje dowolny plik tekstowy z kodem Pseudoassemblera.

Program jako poprawne wejście do interpretacji i symulacji kodu Pseudoassemblera przyjmuje plik tekstowy z kodem Pseudoassemblera spełniającym założenia języka Pseudoassemblera.

Program jako poprawne wejście do symulacji kodu maszynowego z języka Pseudoassemblera przyjmuje plik tekstowy z kodem maszynowym spełniającym założenia kodu maszynowego.

5. Wyjście

Program po zinterpretowaniu kodu w języku Pseudoassemblera na kod maszynowy zapisuje rezultat w pliku o nazwie „[nazwa_pliku_z_kodem_źródłowym]_msck.txt” w folderze pliku źródłowego.

Wynik symulacji, czyli kod maszynowy ze zmodyfikowaną sekcją danych jest zapisywany w pliku o nazwie „[nazwa_pliku_z_kodem_źródłowym]_var.txt” w folderze pliku źródłowego.

Dodatkowo, po ukończeniu symulacji, program wypisuje w konsoli stan końcowy rejestrów oraz komórek pamięci.

6. Uruchomienie

Program można uruchomić z wiersza poleceń.

Pierwszym przyjmowanym argumentem jest względna ścieżka do pliku źródłowego z kodem Pseudoassemblera, bądź kodem maszynowym. W przypadku braku lub niepoprawności ścieżki do pliku źródłowego, program poprosi o jej wpisanie w konsoli uwzględniając resztę argumentów.

Jako drugi argument program przyjmuje „psa_code” lub „msck_code”. W pierwszym przypadku podany plik źródłowy jest interpretowany jako kod Pseudoassemblera, a w drugim jako kod maszynowy. W przypadku nie podania tego argumentu, program uruchamia się w trybie „psa_code”.

Jako trzeci argument program przyjmuje „debug”. Powoduje on uruchomienie symulacji w trybie debugowania. W przypadku nie podania tego argumentu program nie włącza się w trybie debugowania.

Program nie obsługuje sytuacji, gdzie podany jest argument z pominięciem poprzedniego.

Program można włączyć poprzez uruchomienie pliku wykonywalnego. Włącza się on wtedy z argumentami domyślnymi. Program rozpoczyna od poproszenia użytkownika o wpisanie ścieżki względnej do pliku źródłowego. W przypadku niepoprawności ścieżki, program ponownie poprosi o jej wpisanie.

7. Działanie programu

Jeżeli kodem źródłowym jest kod Pseudoassemblera to program wykona analizę poprawności kodu. Następnie zinterpretuje go na kod maszynowy spełniający założenia kodu maszynowego i uruchomi symulację wykonania wynikowego kodu maszynowego.

Jeżeli kodem źródłowym jest kod maszynowy spełniający założenia kodu maszynowego to program od razu uruchomi symulację wykonania kodu maszynowego.

1. Analiza poprawności kodu w języku Pseudoassemblera

Jeżeli napotkany zostanie błąd składniowy, analiza kodu Pseudoassemblera nie jest przerywana, ale nie jest on wtedy interpretowany na kod maszynowy. Po analizie, program kończy swoje działanie i w konsoli wypisuje komunikaty o napotkanych błędach. Jeden komunikat na pierwszy błąd w linii, z 18 możliwych.

Identyfikator błędu	Opis błędu
error_parsing_too_many_arguments	Zbyt dużo argumentów w instrukcji
error_parsing_too_few_arguments	Zbyt mało argumentów w instrukcji
error_parsing_label_starting_with_a_digit	Etykieta rozpoczyna się od liczby
error_parsing_wrong_instruction	Zła nazwa komendy
error_parsing_unrecognised_data_type	Nierozpoznany typ danych
error_parsing_forbidden_whitespaces	Niedozwolone białe znaki
error_parsing_unrecognised_instruction	Zła komenda w danej sekcji
error_parsing_unrecognised_label	Wykorzystanie niezdefiniowanej etykiety
error_parsing_expected_opening_bracket	Oczekiwano nawias otwierający
error_parsing_expected_closing_bracket	Oczekiwano nawias zamykający
error_parsing_expected_comma	Oczekiwano przecinek
error_parsing_expected_asterisk	Oczekiwano gwiazdkę
error_parsing_expected_address	Oczekiwano adres
error_parsing_expected_register	Oczekiwano rejestr
error_parsing_expected_values	Oczekiwano wartość
error_parsing_wrong_address	Zły adres
error_parsing_wrong_register	Zły rejestr
error_parsing_wrong_values	Zła wartość

2. Interpretacja kodu Pseudoassemblera

Program, po wykonaniu analizy kodu źródłowego, interpretuje go na język maszynowy spełniający założenia języka maszynowego.

3. Symulacja wykonania programu

Program wartości niezdefiniowane w kodzie maszynowym ustawia na losowe w przedziale [-RAND_MAX/2, RAND_MAX/2], gdzie w języku C, RAND_MAX jest zdefiniowane na 2^15-1.

Program nie sprawdza czy kod maszynowy spełnia założenia kodu maszynowego, ale jeżeli w czasie symulacji natrafi na ciąg bitów, który nie jest w stanie odczytać to zwraca odpowiedni błąd. Jeżeli program spróbuje wykonać niedozwoloną operację (Dzielenie przez zero, wyjście poza pamięć, przepełnienie), to symulacja zostanie przerwana i użytkownik zostanie o tym poinformowany.

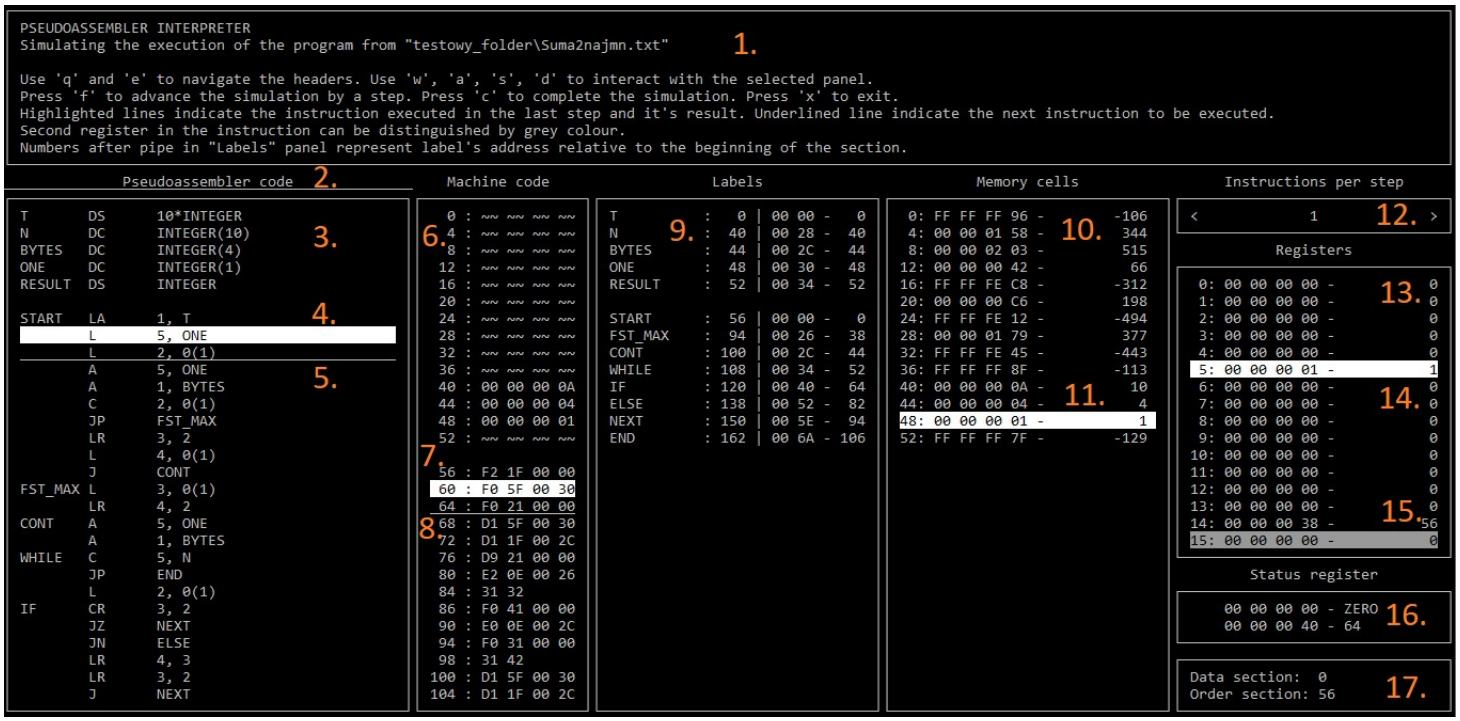
Tryb debugowania umożliwia wizualizację i analizę wykonania symulacji krok po kroku. Jeżeli użytkownik w trybie debugowania wyjdzie w trakcie symulacji, to na wyjściu otrzyma obecny stan programu.

8. Przykładowe programy

- suma2najmn.txt
  Program służący do wyszukiwania dwóch najmniejszych elementów w tablicy i ich zsumowanie.
- merge_sort.txt
  Program służący do scalenia dwóch posortowanych tablic w jedną tablicę posortowaną.
- errtest.txt
  Program służący do przetestowania instrukcji, które powodują błąd.

9. Interfejs trybu debugowania

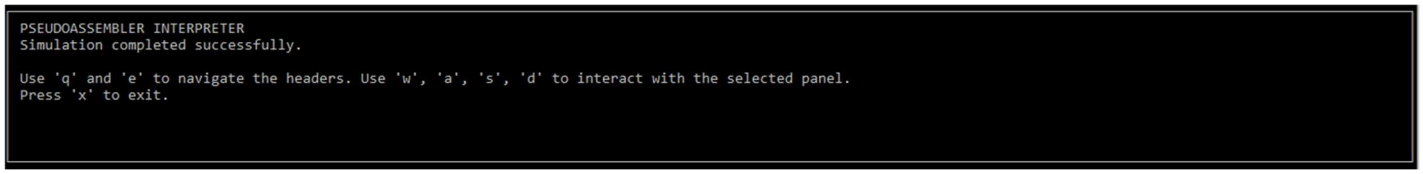
Nawigować po interfejsie można za pomocą klawiszy ‘q’, ‘e’, ‘w’, ‘a’, ‘s’, ‘d’. Klawisz ‘f’ służy do wykonania następnego kroku symulacji. Klawisz ‘c’ służy do ukończenia symulacji. Klawisz ‘x’ służy do przerwania symulacji i wyjścia z programu.



1. Panel informacji. Zawiera instrukcje nawigacji po interfejsie oraz komunikaty od programu.
2. Podkreślenie tytułu panelu. Oznacza aktualnie wybrany panel do interakcji. Można się przemieszczać między „Pseudoassembler Code”, „Machine Code”, „Labels”, „Memory Cells”, a „ Instructions per step” za pomocą klawiszy ‘q’ i ‘e’.
3. Panel „Pseudoassembler Code”. Zawiera źródłowy kod Pseudoassemblera. W przypadku gdy kodem źródłowym jest kod maszynowy, panel jest pusty. Możliwe jest przemieszczanie się po zawartości tego panelu za pomocą klawiszy ‘w’, ‘a’, ‘s’, ‘d’.
4. Podświetlenie linii kodu Pseudoassemblera. Oznacza ono ostatnio wykonaną instrukcję.
5. Podkreślenie linii kodu Pseudoassemblera. Oznacza instrukcję, która zostanie wykonana jako następna.
6. Panel „Machine Code”. Zawiera kod maszynowy po zinterpretowaniu, bądź z pliku źródłowego. Liczby przed dwukropkiem oznaczają adres pierwszego bajtu, który zajmuje w pamięci dana instrukcja. Możliwe jest przemieszczanie się po zawartości tego panelu za pomocą klawiszy ‘w’, ‘s’.
7. Podświetlenie linii kodu maszynowego. Oznacza ostatnio wykonaną instrukcję. Odpowiada ona tej samej instrukcji z panelu kodu Pseudoassemblera.
8. Podkreślenie linii kodu maszynowego. Oznacza instrukcję, która zostanie wykonana jako następna. Odpowiada ona tej samej instrukcji z panelu kodu Pseudoassemblera.

9. Panel „Labels”. Zawiera listę etykiet wykorzystanych w kodzie Pseudoassemblera. Odstęp między blokami etykiet oddziela etykiety z sekcji danych od etykiet z sekcji rozkazów. Liczby po dwukropku oznaczają adres pierwszego bajtu, który zajmuje w pamięci instrukcja, na którą wskazuje etykieta. Liczby po kresce pionowej są kolejno reprezentacją szesnastkową i dziesiętną tego adresu względnego wobec początku sekcji. Możliwe jest przemieszczanie się po zawartości tego panelu za pomocą klawiszy ‘w’, ‘s’.
10. Panel „Memory Cells”. Zawiera listę komórek pamięci symulacji wykonania programu. Liczby przed dwukropkiem oznaczają adres pierwszego bajtu komórki. Liczby po dwukropku to kolejno reprezentacja szesnastkowa i dziesiętna zawartości komórki. Możliwe jest przemieszczanie się po zawartości tego panelu za pomocą klawiszy ‘w’, ‘s’.
11. Podświetlenie komórki pamięci. Wskazuje komórkę pamięci, która została użyta w ostatnio wykonanej instrukcji.
12. Panel „Instructions per step”. Zawiera liczbę instrukcji jaka zostanie wykonana w jednym kroku symulacji. Można tą liczbę zmienić za pomocą klawiszy ‘a’ i ‘d’.
13. Panel „Registers”. Zawiera listę 16 rejestrów symulacji wykonania programu.
14. Podświetlenie rejestru. Wskazuje pierwszy rejestr, który został użyty w ostatnio wykonanej instrukcji.
15. Szare podświetlenie rejestru. Wskazuje drugi rejestr, który został użyty w ostatnio wykonanej instrukcji.
16. Panel „Status register”. Zawiera osiem bajtów rejestru stanu programu. Pierwsze cztery bajty oznaczają znak ostatniej wykonanej operacji, jest to wyrażone słownie po myślniku. Ostatnie cztery bajty oznaczają adres instrukcji, która ma być wykonana jako następna. Liczba po myślniku jest reprezentacją dziesiętną tego adresu.
17. Panel adresów sekcji. Zawiera on reprezentację dziesiętną adresów kolejno sekcji danych oraz sekcji rozkazów.

Program po ukończeniu symulacji wyświetla komunikat o poprawnie ukończonej symulacji.



Program przy natrafieniu na niedozwoloną operację przerywa wykonanie symulacji i wyświetla komunikat o niedozwolonej operacji.