Interpreter Pseudo-assembler'a

Bartosz Zadrożny

Wydział Matematyki i Nauk Informacyjnych
Politechniki Warszawskiej

Rok akademicki 2019/20

1 Wstęp

Niżej opisany program służy do przedstawienia w czasie rzeczywistym działania programów napisanych w języku Pseudo-assembler. Ma on na celu pomóc w nauce danego języka programowania oraz w zrozumieniu działania programów, które zostały przy jego pomocy stworzone.

2 Język Pseudo-assembler

Pseudo-assembler jest uproszczoną wersją jezyka Assembler, który jest symbolicznym zapisem ciągu binarnego opisującego operacje wykonywane na danych. Format komend wygląda następująco:
([etykieta]) [kod rozkazu] [argument 1], ([argument 2])
Elementy podane w nawiasach okrągłych są opcjonalne, natomiast elementy podane w nawiasach kwadratowych są obowiązkowe. Format ten dotyczy jedynie rozkazów, nie obejmuje on dyrektyw alokacji pamięci.

2.1 Uproszczony model pamięci

Operacje w języku Pseudo-assembler korzystają z tak zwanych rejestrów, pamięci opearcyjnej oraz rejestru stanu programu. Oto ich charakterystyka:

- **Rejestry** - jest to szybka podręczna pamięć, która służy do wykonywania operacji arytmetycznych. Dostępnych jest 16 rejestrów ponumerowanych od 0 do 15. Przy czym rejestr 14 jest rejestrem adresowym sekcji danych, a rejestr 15 jest rejestrem adresowym sekcji rozkazów
- **Pamięć operacyjna** - jest to pamięć służąca do przechowywania oraz zapisywania danych

- **Rejestr stanu programu** - służy do zapisywania dodatkowych informacji o dokonywanych obliczeniach, jest aktualizowany po każdym wykonanym rozkazie arytmetycznym i porównawczym

2.2 Rozkazy arytmetyczne

- **A [rejestr 1], [adres komórki pamięci]** - rozkaz dodawania; wynik operacji jest przechowywany w [rejestr 1]
- **AR [rejestr 1], [rejestr 2]** - rozkaz dodawania rejestrów; wynik operacji jest przechowywany w [rejestr 1]
- **S [rejestr 1], [adres komórki pamięci]** - rozkaz odejmowania; wynik operacji jest przechowywany w [rejestr 1]
- **SR [rejestr 1], [rejestr 2]** - rozkaz odejmowania rejestrów; wynik operacji jest przechowywany w [rejestr 1]
- **M [rejestr 1], [adres komórki pamięci]** - rozkaz mnożenia; wynik operacji jest przechowywany w [rejestr 1]
- **MR [rejestr 1], [rejestr 2]** - rozkaz mnożenia rejestrów; wynik operacji jest przechowywany w [rejestr 1]
- **D [rejestr 1], [adres komórki pamięci]** - rozkaz dzielenia; wynik operacji jest przechowywany w [rejestr 1]
- **DR [rejestr 1], [rejestr 2]** - rozkaz dzielenia rejestrów; wynik operacji jest przechowywany w [rejestr 1]

2.3 Rozkazy porównania

- **C [rejestr 1], [adres komórki pamięci]** - rozkaz porównania dwóch wartości; wynik porównania jest przechowywany w rejestrze stanu programu
- **CR [rejestr 1], [rejestr 2]** - rozkaz porównania dwóch wartości przechowywanych w rejestrach; wynik porównania jest przechowywany w rejestrze stanu programu

Wykonanie rozkazu porównania jest identyczne z wykonaniem rozkazu odejmowania, ale wynik odejmowania nie jest zapisywany w rejestrach lub pamięci podręcznej. Jedynie akutalizowana jest wartość rejestru stanu programu w następujący sposób:

- **00**, jeśli wynikiem operacji odejmowania jest 0
- **01**, jeśli wynikiem operacji odejmowania jest liczba dodatnia
- **10**, jeśli wynikiem operacji odejmowania jest liczba ujemna
- **11**, jeśli nastąpił błąd w trakcie wykonywania operacji

2.4 Rozkazy przesyłania

- **L [rejestr 1], [adres komórki pamięci]** - przesyła wartość komórki pamięci do rejestru
- **LR [rejestr 1], [rejestr 1]** - przesyła wartość rejestru drugiego do pierwszego
- **LA [rejestr 1], [adres komórki pamięci]** - przesyła adres komórki pamięci do rejestru
- **ST [rejestr 1], [adres komórki pamięci]** - przesyła wartość rejestru do komórki pamięci o podanym adresie

2.5 Rozkazy skoków

Wszystkie te rozkazy powodują wykonanie rozkazu pod wskazanym adresem, jeśli spełnione są ewentualne warunki.

- **J [adres komórki pamięci]** - skok bezwarunkowy
- **JP [adres komórki pamięci]** - skok, jeśli bity znaku w rejestrze stanu programu wskazują na wartość dodatnią (czyli są postaci 01)
- **JN [adres komórki pamięci]** - skok, jeśli bity znaku w rejestrze stanu programu wskazują na wartość ujemną (czyli są postaci 10)
- **JZ [adres komórki pamięci]** - skok, jeśli bity znaku w rejestze stanu programu wskazują na wartość 0 (czyli są postaci 00)

2.6 Dyrektywy alokacji pamięci

Dyrektywy nie są rozkazami wykonywalnymi; są tylko informacjami dla komputera/interpretera i są one umieszczane na samym początku pliku programu. Format dyrektyw znacząco różni się od formatu rozkazów. W ich przypadku nawiasy okrągłe są częścią składni dyrektywy, nie oznaczają one, że element jest opcjonalny.

- **[etykieta zmiennej] DC INTEGER ([liczba całkowita])** - rezerwuje 4 bajty pamięci i zapisuje na nich podaną liczbę
- **[etykieta zmiennej] DS INTEGER** - rezerwuje 4 bajty pamięci
- **[etykieta zmiennej] DC [liczba komórek]*INTEGER ([liczba całkowita])** - rezeruwje wskazaną liczbę komórek pamięci i inicjalizuje je wskazaną wartością
- **[etykieta zmiennej] DS [liczba komórek]*INTEGER** - rezerwuje wskazaną liczbę komórek pamięci

Przykład dyrektywy alokacji pamięci:
TAB DC 100*INTEGER (20) - nastąpi alokacja 100 komórek pamięci o etykiecie TAB oraz zostaną one zainicjalizowane wartością 20.

2.7 Sposoby adresacji

Pewne rozkazy wykorzystują do pobierania wartości **adres komórki pamięci**. Do adresu możemy się odwoływać na różne sposoby, zależnie od typu rozkazu.

1. **Rozkazy arytmetyczne, porównania i przesyłania**. Sposoby adresacji zostaną zaprezentowane na przykładzie rozkazu przesyłu z pamięci do rejestru. Przy czym zakładamy następującą przykładową sekcję danych:

   TAB1 DS 50*INT
   ET1 DC INT (10)
   TAB2 DC 20*INT (0)
   ET2 DS INT

   (a) **L 2, 216** - przesyłanie zawartości komórki o adresie równym 216 do rejestru o numerze 2

   (b) **L 2, 216(12)** - przesyłanie zawartości komórki o adresie równym sumie 216 i zawartości rejestru 12 do rejestru o numerze 2

   (c) **L 2, ET1** - przesyłanie zawartości komórki opatrzonej etykietą ET1 do rejestru o numerze 2. Etykieta ET1 jest automatycznie zamieniana na 200(14). W rejestrze 14 zapisany jest adres początku sekcji danych

   (d) **L 2, TAB2(5)** - przesyłanie zawartości komórki do rejestru o numerze 2. Adres komórki jest sumą 204, zawartości rejestru 14 i zawartości rejestru 12

2. **Rozkazy skoków**. Sposoby adresacji zostaną zaprezentowane na przykładzie rozkazu skosku bezwarunkowego. Przy czym zakładamy następującą przykładową sekcję rozkazów:

   ```
          J Label
          AR 4, 3
   Label  A 5, ET1
          AR 5, 4
          J Label(3)
   ```

   (a) **J Label** - Etykieta Label zostanie zamieniona na 3(15), a więc ostateczny adres skoku będzie sumą pozycji rozkazu z etykietą Label od początku sekcji rozkazów i zawartości rejestru 15. Rejestr 15 zawiera adres początku sekcji rozkazów

   (b) **J Label(3)** - Skok zostanie wykonany pod adres, który jest sumą pozycji rozkazu z etykietą Label od początku sekcji rozkazów, zawartości rejestru 15 i zawartości rejestru 3

2.8 Przykładowy program w języku Pseudo-assembler

Poniżej zaprezentowany jest przykładowy kod w języku Pseudo-assembler, który oblicza największy wspólny dzielnik dwóch liczb naturalnych.

```
A       DC INTEGER (24)
B       DC INTEGER (6)
WYNIK   DS INTEGER
        L 0, A
        L 1, B
PETLA   CR 0, 1
        JZ WYPISZ
        JN PRAWA
LEWA    SR 0, 1
        J PETLA
PRAWA   SR 1, 0
        J PETLA
WYPISZ  ST 0, WYNIK
```

3 Obsługa interpretera

Aby skorzystać z interpretera, należy uruchomić plik wykonywalny znajdujący się w folderze o nazwie **plik wykonywalny**. Następnie postępować zgodnie z instrukcjami wyświetlanymi na ekranie. W trakcie procesu interpretacji kodu obecnie wykonywana instrukcja jest podświetlana na niebieski kolor w celu ułatwienia śledzenia przebiegu programu. Na niebiesko podświetlona jest różwnież wartość rejestru która uległa zmianie po wykonaniu obecnie podświetlonej instrukcji.

Zostały przygotowane dwa przykładowe programy napisane w języku Pseudo-assembler. Aby interpreter załadował program do wykonania, należy na samym początku podać jego nazwę wraz z rozszerzeniem.

Aby dodać nowy program, należy umieścić go w folderze **programy** w formie pliku tekstowego, a następnie uruchomić ponownie interpreter.

4 Informacje na temat środowiska programistycznego i struktury projektu

Projekt został napisany i zbudowany przy pomocy środowiska Visual Studio 2019. Jest on podzielony modułowo na kilka plików znajdujących się w folderze **pliki źródłowe** w następujący sposób:

- **main.c** - tzw. serce całego programu; znajduje się w nim funkcja main, z poziomu której wywołyane są pozostałe funkcje programu

- **check_functions.h, check_functions.c** - deklaracje i definicje funkcji sprawdzających, wykorzstywanych w trakcie procesu parsowania tekstu programu w Pseudo-assemblerze
- **global_variables.h, global_variables.c** - deklaracje i definicje zmiennych globalnych wykorzystywanych w programie
- **order_functions.h, order_functions.c** - deklaracje i definicje funkcji wykonujących działania odpowiadające rozkazom w Pseudo-assemblerze
- **output_functions.h, output_functions.c** - deklaracje i definicje funkcji odpowiadających za wyświetlanie tekstu na ekranie
- **parsing_and_loading_functions.h, parsing_and_loading_functions.c** - deklaracje i definicje funkcji odpowiadających za import i parsowanie programu napisanego w języku Pseudo-assembler

5 Opis działania interpretera

Poniżej znajduje się uproszczona lista kroków, które wykonuje interpreter w trakcie swojego działania:

1. Ustawienie okna konsoli na całą długość i szerokość ekranu, wypisanie wiadmości powitalnej oraz pobranie nazwy pliku tekstowego zawierającego kod w języku Pseudo-assembler
2. Obliczenie ilości linijek zawierających tekst w programie, a następnie załadowanie każdej linijki tekstu do tablicy
3. Parsowanie linijek, które są dyrektywami alokacji pamięci, obliczanie ilości potrzebnego miejsca w pamięci oraz jego dynamiczna alokacja. Zmienne zadeklarowane w programie w języku Pseudo-assembler są reprezentowane jako struktury zawierające ich etykietę, rozmiar oraz wskaźnik do miejsca w zaalokowanej dynamicznie pamięci, które przechowuje jej wartość
4. Obliczenie ilości rozkazów znajdujących się w programie, a następnie dynamiczna alokacja pamięci potrzebnej do przechowania ich w programie. Rozkaz jest reprezentowany przez strukturę zawierającą pola odpowiadające etykiecie, rozkazowi, argumentowi pierwszemu i drugiemu. Następnie następuje parsowanie każdego z rozkazów i uzupełnianie tablicy struktur odpowiednimi wartościami
5. Wykonywanie rozkazów poprzez przeprowadzanie operacji im odpowiadających. Przy czym jednocześnie na ekranie wyświetlany jest stan rejestrów, zadeklarowanej pamięci oraz aktualnie interpretowany kod w języku Pseudo-assembler
6. Zwolnienie dynamicznie zaalokowanej pamięci i zakończenie pracy programu

6 Szczególne uwagi i znane błędy

Interpreter może nie działać poprawnie jeżeli pomiędzy sekcją deklaracji alokacji pamięci, a sekcją rozkazów jest odstęp w postaci jednej lub wielu pustych linii. W trakcie testowania nie wpłynęło to na ostateczny wynik programów, ale uprasza się o brak odstępu pomiędzy wyżej wymienionymi sekcjami.