­­­­PSEUDO - ASSEMBer INTERPRETER

Author: Dawid Kożykowski

# Manuals and Documentation

### Table of contents:

### [1. Basic informations](#_Basic_informations) [2. Manuals](#_manuals) [2.1 Structure of language](#_Structure_of_language) [2.2 Addressing](#_Addressing) [2.3 Allocating memory commands](#_Allocating_memory_commands_2) [2.4 Operating commands](#_Operating_commands) [2.5 Terminal window](#_Terminal_window) [2.6 Getting started](#_2.6__Getting) [3. Documentation](#_3._documentation) 3.1 3.2

# 1. Basic informations

|  |  |
| --- | --- |
|  | This program is an interpreter for the pseudo-assembly language designed by prof. Homenda. The program reads list of commands with arguments from a file and lets user see changes made by every single command. This project is licensed under MIT License. |

# 2. manuals

## Structure of language

|  |  |
| --- | --- |
|  | It is assumed, that there are 15 registers user can use, 2 different types of commands (allocating memory or operating commands) and 4 different program states: „00” – idle state „01” – positive state „10” – negative state “11” – error. First block of commands is obligated to be allocating memory type. After first usage of operating command there should NOT be a single memory allocation command left. Every command in this language looks as follows:  “<label> <type> <argument1>,<argument2>”, e.g. “FOUR DC 1,4” |

## Addressing

|  |  |
| --- | --- |
|  | There are few different ways of addressing. These are their interpretations:  <4096> (e.g.: A 1, 4096) – 4 byte cell with address 4096 <4096(12)> (e.g.: A 1, 4096(12)) – 4 byte cell with address (4096 + address stored in  12th register) <TAB> (e.g.: A 1, TAB) – 4 byte cell with address stored under label “TAB” <TAB(12)> (e.g.: A 1, TAB(12)) – 4 byte cell with address stored under label “TAB” increased by address stored in 12th register) |

## Allocating memory commands

|  |  |
| --- | --- |
|  | Allocating memory commands are placed at the beginning of program. All of them adds some number with or without value to program memory. Labels are necessary. These are all of them with interpretations:  <label> DC INTEGER(<arg1>) - add a single number named <label> with value <arg1> <label> DC <arg1>\*INTEGER – add an array with <arg1> cells named <label> <label> DC <arg1>\*INTEGER(<arg2>) – add an array with <arg1> cells named <label>, each cell has value <arg2> <label> DS INTEGER - add a single number named <label> <label> DS <arg1>\*INTEGER - add an array with <arg1> cells named <label> |

## Operating commands

|  |  |
| --- | --- |
|  | Operating commands are placed after allocating memory ones. They are used to do calculations, comparisons etc. Labels are not necessary. These are all of them with interpretations:   <label> A <arg1>, <arg2> - adds value stored in cell with address <arg2> to register with number <arg1> <label> AR <arg1>, <arg2> - adds value stored in register with number <arg2> to register with number <arg1> <label> S <arg1>, <arg2> - subtracts value stored in register with number <arg1> by value stored in cell with address <arg2>  <label> SR <arg1>, <arg2> - subtracts value stored in register with number <arg1> by value stored in register number <arg2> <label> M <arg1>, <arg2> - multiplies value stored in register with number <arg1> by value stored in cell with address <arg2> <label> MR <arg1>, <arg2> - multiplies value stored in register with number <arg1> by value stored in register number <arg2> <label> D <arg1>, <arg2> - divides value stored in register with number <arg1> by value stored in cell with address <arg2> <label> DR <arg1>, <arg2> - divides value stored in register with number <arg1> by value stored in register number <arg2> <label> C <arg1>, <arg2> - compares value stored in register with number <arg1> with value stored in cell with address <arg2>, sets state: “00” if they are equal, “01” if value in arg1 is greater than value in arg2, “10” in different case <label> CR <arg1>, <arg2> - compares value stored in register with number <arg1> with value stored register number <arg2>, sets state: “00” if they are equal, “01” if value in arg1 is greater than value in arg2, “10” in different case <label> J <arg1> - moves to line with label equal to arg1 <label> JP <arg1> - if program state is equal to “01” moves to line with label equal to arg1 <label> JN <arg1> - if program state is equal to “10” moves to line with label equal to arg1 <label> JZ <arg1>, <arg2> - if program state is equal to “00” moves to line with label equal to arg1 <label> L <arg1>, <arg2> - sets value stored in register with number <arg1> to value stored in cell with address <arg2> <label> LA <arg1>, <arg2> - sets value stored in register with number <arg1> to arg2 <label> LR <arg1>, <arg2> - sets value stored in register with number arg1 to value stored in register with number arg2  <label> ST <arg1>, <arg2> - sets value stored in cell with address arg2 to value stored in register with number <arg1> |

## Terminal window

|  |  |
| --- | --- |
|  | The green line is the commands section is a line of program that has just been executed.    Red lines in memory or register section shows differences made by current line of code |
|  |  |

## 2.6 Getting started

|  |  |
| --- | --- |
|  | To get started just simply make a txt file called „commands” in the same folder that You put interpreter.c , write lines of Your program in this text file, compile interpreter.c and run it. |

# 3. documentation

## Procesy biznesowe lub systemy objęte wpływem projektu

|  |  |
| --- | --- |
|  | Separeted GUI code and Back End code |

## Specjalne wykluczenia z zakresu

|  |  |
| --- | --- |
|  |  |

## Plan wdrożenia

|  |  |
| --- | --- |
|  |  |

## Ogólny harmonogram lub oś czasu

|  |  |
| --- | --- |
|  |  |