Virtual Machine

1.0

Generated by Doxygen 1.9.1

Chapter 1

Virtual_Machine

A simple virtual machine created from scratch written in c++

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 VM::assembler Class Reference

The assembler class for Virtual Machine. This class provides functionality to assemble instructions and write them to a binary file.

`#include <assembler.h>`

Public Member Functions

- assembler ()=default

  Default constructor for the assembler class.
- assembler (std::string &fname)

  Parameterized constructor for the assembler class.
- assembler (const char *fname)

  Parameterized constructor for the assembler class.
- std::vector< uint32_t > compile_instructions (std::vector< std::string > &str)

  Compiles a vector of assembly instructions into machine code.
- std::FILE * getFile ()

  Gets the file pointer of the assembled binary file.
- std::string getFileName ()

  Gets the name of the file being assembled.
- uint32_t map_to_intstuction (char &c)

  Maps a character to its corresponding instruction in machine code.
- void assemble_to (const char *fname)

  Writes the assembled instructions to a specified file.

4.1.1 Detailed Description

The assembler class for Virtual Machine. This class provides functionality to assemble instructions and write them to a binary file.

Definition at line 18 of file assembler.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 assembler() [1/2]

```
VM::assembler::assembler (
            std::string & fname )
```

Parameterized constructor for the assembler class.

Parameters

fname	Input file.

Definition at line 10 of file assembler.cpp.

```
10                                      {
11     //create input files
12     std::ifstream file;
13     file.open(fname);
14
15     if(file.is_open())  throw std::invalid_argument("File not found") ;
16
17     std::string line;
18     std::string buf;
19     while (std::getline(file, line)) buf += line + "\n";
20
21     file.close();
22
23     //parse the file
24     VM::lexer lex;
25     std::vector<std::string>tokens = lex.lex(buf);
26     std::vector<uint32_t> instructions = compile_instructions( tokens);
27
28
29     //write to output
30     std::ofstream output;
31     output.open(fname + ".out.bin", std::ios::binary); //output file in binary format
32
33     for (uint32_t i = 0; i < instructions.size(); i++) {
34         output.write(reinterpret_cast<char *>(&instructions[i]), sizeof(uint32_t));
35     }
36     output.close();
37
38 }
```

4.1.2.2 assembler() [2/2]

```
VM::assembler::assembler (
            const char * fname )
```

Parameterized constructor for the assembler class.

Parameters

fname	Input file .

Definition at line 40 of file assembler.cpp.

```
40                                      :m_fname{fname}{
```

```
41      std::string sfname{m_fname};
42      sfname += ".out.bin";
43
44
45      //create input files
46      std::ifstream file;
47      file.open(fname);
48
49      if(!file.is_open())  throw std::invalid_argument("File not found") ;
50
51      std::string line;
52      std::string buf;
53      while (std::getline(file, line)) buf += line + "\n";
54
55      file.close();
56
57      //parse the file
58      VM::lexer lex;
59      std::vector<std::string>tokens = lex.lex(buf);
60      std::vector<uint32_t> instructions = compile_instructions( tokens);
61
62
63      //output
64      m_file = std::fopen(sfname.data(), "wb");
65
66
67
68      for (uint32_t i = 0; i < instructions.size(); i++) {
69          // Write each element of the vector to the file
70          std::fwrite(&instructions[i], sizeof(uint32_t), 1, m_file);
71      }
72
73      std::fclose(m_file);
74  }
```

4.1.3 Member Function Documentation

4.1.3.1 assemble_to()

```
void VM::assembler::assemble_to (
            const char * fname )
```

Writes the assembled instructions to a specified file.

Parameters

fname	The name of the file to write the assembled instructions to.

4.1.3.2 compile_instructions()

```
std::vector< uint32_t > VM::assembler::compile_instructions (
            std::vector< std::string > & str )
```

Compiles a vector of assembly instructions into machine code.

Parameters

str	Vector of strings containing assembly instructions.

Returns

Vector of 32-bit unsigned integers representing compiled instructions.

Definition at line 78 of file assembler.cpp.

```
78                                                                                {
79      std::vector<uint32_t> instructions;
80      //go through each line
81      for(size_t  i = 0; i < str.size(); ++i){
82
83          if( isInt(str[i]) )  instructions.push_back(std::stoi(str[i]));
84          else{ //go through each char in the string
85              for(char c: str[i]){
86              uint32_t instruction = map_to_intstuction(c);
87               instructions.push_back(instruction);
88              }
89
90          }
91      }
92      instructions.push_back(0x40000000); // add halt at the end
93
94      return instructions;
95
96 }
```

4.1.3.3 getFile()

```
std::FILE * VM::assembler::getFile ( )
```

Gets the file pointer of the assembled binary file.

Returns

File pointer to the assembled binary file.

Definition at line 124 of file assembler.cpp.

```
124 {return m_file; }
```

4.1.3.4 getFileName()

```
std::string VM::assembler::getFileName ( )
```

Gets the name of the file being assembled.

Returns

The name of the file being assembled.

Definition at line 125 of file assembler.cpp.

```
125 {return m_fname + ".out.bin";}
```

4.1.3.5 map_to_intstuction()

```
uint32_t VM::assembler::map_to_intstuction (
            char & c )
```

Maps a character to its corresponding instruction in machine code.

Parameters

c	The character to be mapped.

Returns

The machine code instruction corresponding to the input character.

Definition at line 106 of file assembler.cpp.

```
106                                                 {
107
108
109     switch(c){
110         case PLUS: return 0x40000001;
111         case MINUS: return 0x40000002;
112         case MULTI: return 0x40000003;
113         case DIVIDE: return 0x40000004;
114
115                 //pray to whatever god you want  we don't get this
116         default: return -1;
117
118     }
119
120     return -1;
121
122 }
```

The documentation for this class was generated from the following files:

- include/assembler.h
- src/assembler.cpp

4.2 VM::lexer Class Reference

Public Member Functions

- std::vector< std::string > **lex** (std::string s)

4.2.1 Detailed Description

Definition at line 5 of file lexer.h.

The documentation for this class was generated from the following files:

- include/lexer.h
- src/lexer.cpp

4.3 VM::Machine Class Reference

Machine class that initializes the virtual machine and runs the binary file as it's input.

```
#include <Machine.h>
```

Public Member Functions

- Machine ()

  Constructor that allocates memory.
- void run ()

  Run function to start the machine, provided it's loaded a program,.
- void loadProg (std::vector< uint32_t > prog)

  load Program into memory
- void loadProg (int argc, char *argv[])

  load Program into memory. This function loades the instructions as command line arguments.

4.3.1 Detailed Description

Machine class that initializes the virtual machine and runs the binary file as it's input.

Definition at line 20 of file Machine.h.

4.3.2 Member Function Documentation

4.3.2.1 loadProg() [1/2]

```
void VM::Machine::loadProg (
            int argc,
            char * argv[] )
```

load Program into memory. This function loades the instructions as command line arguments.

Parameters

argc	argument count
argv[]	array of arguments

Definition at line 88 of file Machine.cpp.

```
88                                                  {
89     for (int32_t i = 0; i < argc ; ++i) {
90         std::cout << "Read" << "\n";
91
92         memory[pc + i] = std::atoi(argv[i]);
93
94     }
95 }
```

4.3.2.2 loadProg() [2/2]

```
void VM::Machine::loadProg (
            std::vector< uint32_t > prog )
```

load Program into memory

Parameters

prog	array of instructions (in binary format) to be loaded into memory

Definition at line 82 of file Machine.cpp.

```
82                                                  {
83
84     for (int32_t i = 0; i < static_cast<int32_t>( prog.size()); ++i) {
85         memory[pc + i] = prog[i];
86     }
87 }
```

The documentation for this class was generated from the following files:

- include/Machine.h
- src/Machine.cpp

4.4 RNP Class Reference

Public Member Functions

- strings **rnp** (std::string s)

4.4.1 Detailed Description

Definition at line 23 of file RNP.h.

The documentation for this class was generated from the following file:

- RNP_Lang/RNP.h

Chapter 5

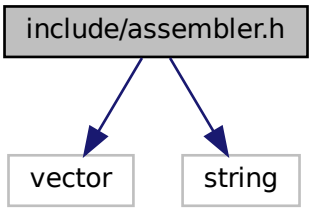
File Documentation

5.1 include/assembler.h File Reference

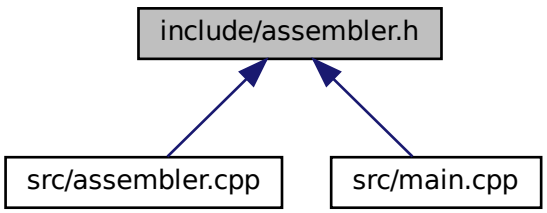
Assembler file for transforming the input file into instructions.

```
#include <vector>
#include <string>
```

Include dependency graph for assembler.h:



This graph shows which files directly or indirectly include this file:



Classes

- class VM::assembler

  The assembler class for Virtual Machine. This class provides functionality to assemble instructions and write them to a binary file.

5.1.1 Detailed Description

Assembler file for transforming the input file into instructions.

Author

Aleksandar Dikov (528052)

Version

Date

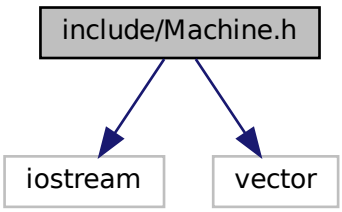
2024-01-20

5.2 include/Machine.h File Reference

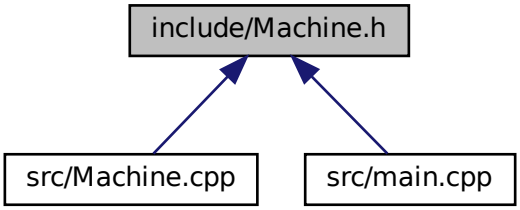
Machine class for initializind the Virtual machine.

```
#include <iostream>
#include <vector>
```

Include dependency graph for Machine.h:



This graph shows which files directly or indirectly include this file:



Classes

- class VM::Machine

  Machine class that initializes the virtual machine and runs the binary file as it's input.

5.2.1 Detailed Description

Machine class for initializind the Virtual machine.

Author

Aleksandar Dikov (528052)

Version

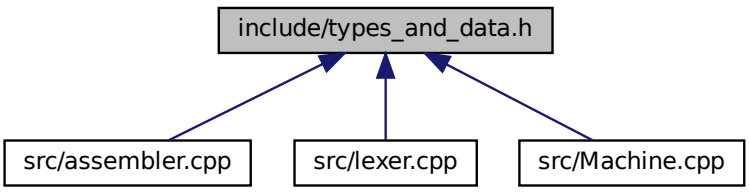
Date

2024-01-22

5.3 include/types_and_data.h File Reference

Types and Data.

This graph shows which files directly or indirectly include this file:



Enumerations

- enum { POS_INT = 0 , INSTRUCTION , NEG_INT }
- enum { PLUS = '+' , MINUS = '-' , MULTI = '∗' , DIVIDE = '/' }

  Enumeration representing arithmetic operations. This is Used by assembler.h to parse the syntax of the input file into instructions for the machine.
- enum {
  HALT = 0 , ADD , SUB , MUL ,
  DIV , OP_AND , OP_OR , LOD }

  Enumeration for virtual machine instructions. This is used by machine.h to preform logical operations for the calculator.
- enum State : char {
  START , READCHAR , READBLOCK , SKIP ,
  DUMP , END , **START** , **READCHAR** ,
  READBLOCK , **DUMP** , **END** }

  Enumeration representing states in the state machine. This is used by lexer.h for converting the bytes from the input file into a binary format for the virtual machine.

5.3.1 Detailed Description

Types and Data.

Author

Stiliyan Batinkov (530007)

Version

1.2

Date

2024-01-18

This class holds the values for the different data types, instructions and states. This file is used by all other header files (Machine.h, assembler.h, lexer.h)

5.3.2 Enumeration Type Documentation

5.3.2.1 anonymous enum

`anonymous enum`

Enumerator

POS_INT	Positive integer.
INSTRUCTION	Instruction.
NEG_INT	Negative integer.

Definition at line 19 of file types_and_data.h.

```
19      {
20      POS_INT = 0,
21      INSTRUCTION,
22      NEG_INT
23  };
```

5.3.2.2 anonymous enum

```
anonymous enum
```

Enumeration representing arithmetic operations. This is Used by assembler.h to parse the syntax of the input file into instructions for the machine.

Enumerator

PLUS	Addition operation.
MINUS	Subtraction operation.
MULTI	Multiplication operation.
DIVIDE	Division operation.

Definition at line 29 of file types_and_data.h.

```
29      {
30      PLUS = '+',
31      MINUS = '-',
32      MULTI = '*',
33      DIVIDE = '/'
34  };
```

5.3.2.3 anonymous enum

```
anonymous enum
```

Enumeration for virtual machine instructions. This is used by machine.h to preform logical operations for the calculator.

Enumerator

HALT	Halt
ADD	Addition
SUB	Subtraction
MUL	Multiplication
DIV	Division
OP_AND	Logical AND
OP_OR	Logical OR
LOD	Load

Definition at line 41 of file types_and_data.h.

```
41       {
42     HALT = 0,
43     ADD,
44     SUB,
45     MUL,
46     DIV,
47     OP_AND,
48     OP_OR,
49     LOD
50 };
```

5.3.2.4 State

```
enum State :  char
```

Enumeration representing states in the state machine. This is used by lexer.h for converting the bytes from the input file into a binary format for the virtual machine.

Enumerator

START	Initial state.
READCHAR	Reading characters.
READBLOCK	Reading blocks of characters.
SKIP	Skipping characters.
DUMP	Dumping information.
END	End state.

Definition at line 56 of file types_and_data.h.

```
56           : char {
57     START,
58     READCHAR,
59     READBLOCK,
60     SKIP,
61     DUMP,
62     END
63 };
```

Index