Intermediate code and its interpreter

The virtual processor

The virtual processor has three registers:

- Program counter PC
- The stack pointer SP, pointing to the last occupied byte on the stack, the stack grows in the direction of decreasing addresses
- Frame pointer BP

The virtual processor has the memory-memory architecture, i.e. all instructions operate directly on the operands in memory. The data and program memory are separated.

The single instruction has the following form:

label: mnemonic.s arg1, arg2, arg3; comment

The label is optional. The type-of- arguments marker s and arguments arg1, arg2 and arg3 are not present in all instructions. The text after the semicolon till the end of the line is treated as comment.

There are three addressing modes:

- Immediate addressing, denoted by # as the first character of address expression
- Direct addressing, characterized by the lack of any special character before the address expression
- Indirect addressing, denoted by * as the first character of the address

The address expression can have one of the following forms:

- label
- integer
- real (only in immediate mode)
- bp + integer
- bp - integer
- bp

The argument type marker can have one of the two values:

- i for integer type
- r for real type

In case of conditional jumps, the last argument (address) is always of integer type.
The label is a sequence of alphanumeric characters starting with the letter.

For example, the instruction

add.r #3.1415926, bp+18, *bp-32

adds the value of 3.1415926 to the real number stored at the address BP+18 and places the result at the address read from the memory location, which address is located at BP-32.

The following instructions are recognized:

mov.s arg1, arg2

Copies data of type s from arg1 to arg2

add.s arg1, arg2, arg3

Adds arg1 to arg2 and places the result in arg3

call arg1

Places at the top of the stack the address of the next instruction and moves arg1 to PC

enter arg1

Sets the value of frame pointer BP and reserves the place on the stack for the local variables. It is equivalent to the following sequence of operations:

push BP
BP:=SP
SP:=SP-arg1

leave

Restores the state of the stack, which existed before execution of **enter** instruction. It is equivalent to the following sequence of operations:
SP := BP
pop BP

return

Pops an integer from the stack and moves it to the program counter register PC

write.s arg1

Prints the value of arg1 to the standard output

push.s arg1

Pushes the value of arg1 to the stack

inscp arg1

Increments the stack pointer SP by arg1

jump arg1

Assigns the program counter PC with the value of arg1

je.s arg1, arg2, arg3

Compares arg1 with arg2, if the numbers are equal, moves arg3 to PC, otherwise increments PC by 1

jge.s arg1, arg2, arg3

Compares arg1 and arg2 as the signed numbers, if arg1 is greater or equal than arg2, moves arg3 to PC, otherwise increments PC by 1

exit

Ends the program execution

The structure of the program

The program consists of two basic parts: the assembler and the processor emulator.
The assembler processes the file given as its first argument. The assembly has two passes, in the first one the addresses of labels are computed and stored in the symbol table **symtab**, in the second one instructions are stored in a vector named **instructions**. The pass number (0 or 1) is stored in a global variable **pass**. The function **analyze** is responsible for processing of the individual instructions. This function accepts as an argument the preprocessed line of the program, with the comment removed and all characters converted to lowercase. During analysis of the program the following data structures are used: the association table **opcode_table**, which contains names and codes of all instructions, the vector **default_argtype_table**, storing the default argument types for all instruction and the vector **syntax_table**, storing the addresses of procedures checking the number and types of arguments of all instructions. The above data structures are filled out by the function **setup_opcodes**, called at the beginning of function **main**, based on the contents of the array **opcodes**. The function **extract_label** is responsible for analysis of labels, **extract_instr** for analysis of instruction mnemonics, and **extract_address** for analysis of expressions denoting a single argument. The above functions together with some helper functions are stored in a file *analyzer.cpp*. The syntax errors are signaled by throwing an exception **syntax_error**. Its constructor accepts as an argument the text string describing the error in more details.

The processor emulator is contained in a file *machine.cpp*. The function **execute** reads the instructions from the array **instructions** and calls the function responsible for their emulation via the array **dispatch_table**, filled out in a function **setup_opcodes**. Execution of the program finishes after the instruction **exit** is encountered. The names of functions responsible for emulation of instructions start with the prefix **h_**. These functions use extensively the helper functions **get_int_operand** and **get_real_operand** returning the values of the source instruction arguments and **get_int_operand_ref** and **get_real_operand_ref**, returning the reference to the target argument. In case of an attempt of access of a nonexistent memory location or execution of incorrect instruction, the **segmentation_fault** exception is thrown. This file contains also a definition of series of functions with names prefixed with **s_**, used in the assembly phase, responsible for checking the number and types of instruction arguments. The address of an array emulating the data memory is stored in a global variable **memory**, the size of data memory is stored in a global variable **memorysize**. Before the emulation begins, the program counter PC is set to 0, and the stack pointer SP to **memorysize**.
The header file *vm.h* contains, among others, declarations of various types. The instruction code is determined by the enumeration type **opcode**, the value OC_FINAL must be always at the end of the list and denotes the number of different instructions recognized by the emulator. The enumeration type **argtype** denotes the type of operand, **addrmode** denotes the immediate, direct or indirect addressing mode, **basereg** indicates if the frame pointer register BP is used in the addressing. Types INT, UINT and REAL determine the data types used by the emulator.