**Заводов Андрей Павлович (P3119)**

**Спецификация виртуальной машины**

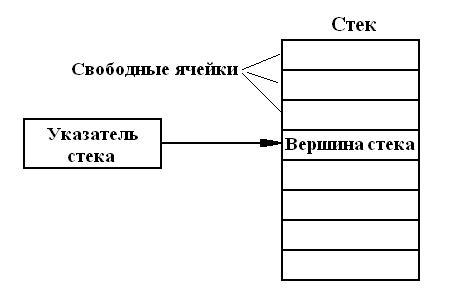
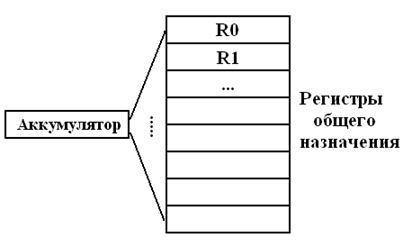
**«Edem-VM»**

**1. Общие сведения**

**Edem-VM** – простейшая виртуальная машина, предназначенная для работы с целыми числами и символами. Предусматривает операторы ветвления, циклы и массивы размерностью до 256 полей.

**2. Модель вычислений**

Данная VM может использовать как стековую модель вычислений, так регистровую. Инструкции для работы с ними приведены ниже.



**3. Модель памяти**

Данные и промежуточные результаты будут опционально хранится либо в стеке, либо в одном из 256 регистров, доступных для хранения данных.

**4. Набор инструкций**

**4.1. Для работы со стеком**

**Add <value> –** добавление в стек **value**.

**Print** - вывод верхнего элемента стека на экран.

**Pop <reg> -** копирование верхнего элемента стека в регистр **reg**.

**Sum –** сложение двух верхних элементов и запись результата обратно в стек.

**Pow** - умножение двух верхних элементов и запись результата обратно в стек.

**Div** – целочисленное деление второго элемента на первый и запись результата обратно в стек

**Diff** – вычитание из второго элемента первого и запись результата обратно в стек.

**4.2. Для работы со регистрами**

**RAdd <reg><value> -** записывает значение **value** в регистр по номеру **reg**.

**RPrint <reg> -** выводит на экран значение регистра по номеру **reg**.

**RSum <reg-1><reg-2><reg-3> -** суммирует значения в регистрах **reg-1**, **reg-2** и записывает результат в **reg-3**.

**RPow <reg-1><reg-2><reg-3> -** перемножает значения в регистрах reg-1, reg-2 и записывает результат в reg-3.

**RDiv <reg-1><reg-2><reg-3> -** вычисляет остаток от деления **reg-1** на **reg-2** и записывает результат в **reg-3**.

**RDiff <reg-1><reg-2><reg-3> -** вычитает значение регистра **reg-2** из **reg-1** и записывает результат в **reg-3**.

**4.3. Организация ветвлений**

**IF <** **condition>**

**Operator-true**

**Operator-false**

Выше представлена схема ветвлений. Если **condition=true** выполняется инструкция на следующей стоке за условием. Если **condition=false** выполняется инструкция через строку от условия.

**4.4. Организация циклов**

**FOR (variable; start; finish)**

**Operator**

Переменная **variable** принимает все значения от **start** до **finish**. На каждой итерации цикла выполняется инструкция **operator**. **Variable** может изменяться в теле цикла, но шаг цикла всегда равен 1.

**4.5. Специальные символы**

**// -** строка после этого символа не читается компилятором.

**#<value>** - распознавать **value**, как код символа.

**4.6. Работа с файлами**

**AlgRead <path> -** чтение алгоритма из файла по пути **path**

**ValRead <path><reg>** - сохраняет значение из регистра **reg** по пути **path**.

**ValWrite <path><reg>** - читает файл по пути **path** и сохраняет первое корректное значение в регистр **reg**.

**StWrite <path>** - сохраняет верхнее значение стека.

**5. Режим адресации**

**Непосредственная адресация –** данные для инструкции входят в саму инструкцию.

Загрузка и получение данных производится с помощью специальных инструкций.

Данные могут быть заданы как непосредственно в самом коде алгоритма, так и прочитаны из файла.

**6. Форма записи алгоритма**

Алгоритм подается на стандартный поток чтения либо загружается из файла целиком (не по 1 инструкции). Инструкции располагаются каждая на новой строке. Если компилятор обнаруживает ошибку в инструкции или инструкция ему не известна, исполнение алгоритма не завершается, но выводится сообщение об ошибки. Данная строка в свою очередь просто игнорируется исполнителем.