**BC/NW 2016 № 2 (29): 3.1**

**РАЗРАБОТКА ПРОГРАММНОГО ЭМУЛЯТОРА**

**БАЗОВОЙ РЕГИСТРОВОЙ МАШИНЫ**

Луковцев Н.С.

Процессом обработки данных (ПОД) называется процесс ввода, преобразования, возможно, наполнения и вывода данных, обеспечивающий решение определенной задачи или множества взаимосвязанных задач.

Для того чтобы ПОД мог быть выполнен, его необходимо задать, используя языки программирования. Однако для изучения организации ОД и архитектур СОД такие языки слишком громоздки, их синтаксис, а особенно семантика трудно формально определить. Необходимо, чтобы способ описания вычисления предлагал простой, фундаментальный язык.

В качестве модели обработки данных рассмотрим простейшую вычислительную модель – базовую регистровую машину(БРМ). БРМ это линейно упорядоченное множество регистров, хранящих натуральные числа, для которых определены три операции: обнуление, инкремент и переход по равенству.

Целью разработки является создание программного эмулятора БРМ. Эмулятор должен строиться как интерпретатор внутреннего языка БРМ – машинных команд, имеющих двоичное представление. Это представление получается в результате трансляции исходного кода программы. Эмулятор также должен предоставлять пользователю возможность пошаговой отладки программы.

**Модели обработки данных**

С понятием ОД неразрывно связаны два понятия:

1. Система (или механизм), реализующая процесс обработки данных.

2. Пользователь, в интересах которого решаются эти задачи.

Основные свойства, которые присуще ПОД именно как процессу:

* степень параллелизма: последовательный, параллельный;
* степень детерминизма;
* ручной, автоматизированный.

В общем классе ПОД есть подкласс, который называется процессом вычислений (ВП).

Процесс вычислений – это самозавершаемый процесс ОД, после того, как получены все необходимые результаты преобразования.

Функция наполнения была в определении со словами возможно. Она не относится к вычислению. Вычисление не связано с наполнением. В процессе вычисления данные преобразуются, и результат выдается пользователю.

Обработка данных – это самонезавершаемый процесс. Как правило, она связана с накоплением.

Для того, чтобы ПОД мог быть выполнен, его необходимо задать. Чтобы задать вычисления нужно средство описания ПОД, которое позволяет обеспечить:

1. синтаксическое представление алгоритмов (язык описания ПОД) – {набор команд, форматы команд и т.д.} или способ задания правильных конструкций.
2. способ получения результата по записи алгоритма (интерпретатор, обрабатывающий конструкции языка).

Собственно, процесс выполнения конструкции языка интерпретатором и есть процесс вычисления.

Определим модель обработки данных как способ задания процессов обработки данных при помощи языка, который обеспечивал бы формальное описание его интерпретатора, т.е. его структуры и алгоритма функционирования.

Требования к языку – его легко можно интерпретировать.

Требования, предъявляемые к модели:

1. модель должна быть полна в некотором классе алгоритмов (полнота модели);
2. фундаментальность – модель должна ясно, просто и четко специфицировать и должна иметь широкую и строгую семантику;
3. модель должна быть операционной.

Таким образом, модель, задающая процесс вычисления, содержит 3 составляющих:

M<L, S, F>

<S, F> – абстрактная машина.

1. L – язык описания ПОД (язык модели).
2. Абстрактная машина (АМ) – способ показать, как алгоритм, записанный на языке L, применяется к ИД для получения результатов.

S – структура АМ: совокупность элементов хранения данных, т.е. элементов, в которых может храниться любая конструкция языка L (состояние элементов определяется через конструкции языка L).

F – алгоритм выполнения языка L на структуре S (алгоритм функционирования АМ).

Наше определение модели обработки данных подходит под операционное уточнение алгоритма, в котором определяется множество элементов хранения данных и специфицируется порядок изменений состояния элементов хранения данных под воздействием изменения состояния определенных RG.

Действительно: F – множество правил, которые показывают, как изменяется состояние АМ при интерпретации ею тех или иных конструкций языка L.

Рассмотрим в качестве модели обработки данных простейшую вычислительную модель – БРМ. С точки зрения определения модели удобнее поменить порядок рассмотрения элементов тройки M < L, S, F >

1. S – структура определяет множество элементов хранения данных в АМ (регистры).
2. L – язык – определение правильных синтаксических конструкций (команды, операторы).
3. F – алгоритм выполнения этих конструкций, показывающий преобразование информации, хранящейся в S.

Структура машины S состоит из регистров двух назначений, способных хранить натуральные числа: регистр данных, счетчик команд (регистр, содержащий адрес инструкции, которая будет выполняться).

L: программа – пронумерованная последовательность команд трех типов:

* очистка регистра;
* увеличение на 1 содержания регистра;
* переход по равенству содержимого двух регистров.

F – необходимо определить элементы функционирования F в терминах S: что делает АМ при выполнении конкретной команды; как изменится состояние АМ.

Разработка эмулятора именно для этой БРМ будет описана далее.

**Подготовка к разработке программного эмулятора**

***Синтаксис и семантика команд БРМ***

Программа может использовать 3 базовые команды: ZERO, INCR, BRAN.

* Команда ZERO – команда обнуления содержимого указанного регистра.

Синтаксис: ZERO\_R<номер регистра>

Семантика команды: 0Ri;(IP)+1IP, где ( ) – содержимое регистра

* Команда INCR – команда увеличения на 1 содержимого указанного регистра.

Синтаксис: INCR\_R<номер регистра>

Семантика команды: (Ri)+1Ri; (IP)+1IP

* Команда BRAN – команда перехода к инструкции, номер которой указан в команде по равенству содержимого двух регистров.

Синтаксис: BRAN\_R<номер регистра>,R<номер регистра>,<номер инструкции>

Семантика команды: если (Ri)=(Rj), то aIP, иначе (IP)+1IP

***Внутреннее представление кода***

Эмулятор должен строиться как интерпретатор внутреннего языка БРМ – машинных команд, имеющих двоичное представление. Это представление получается в результате трансляции исходного кода программы, описанным ниже способом.

Код операции команды INCR – 30h;

Код операции команды ZERO – 50h;

Код операции команды BRAN – 70h;

Внутреннее представление каждой из команд INCR и ZERO состоит из одного байта. Старшие 4 бита – код операции команды, младшие 4 – это номер регистра (т.к. в программе 16 регистров, четырех бит достаточно).

Например, внутреннее представление команды INCR R7 будет 37h, а внутреннее представление команды ZERO R11 будет 5Bh. Программа будет показывать результат трансляции в 16-ной системе для удобства при проверке и экономии места.

Внутреннее представление команды BRAN состоит из 4 байтов. Старшие 4 бита первого байта – код операции команды BRAN, младшие 4 – номер первого регистра; старшие 4 бита второго байта – операционный код регистра, младшие 4 – номер второго регистра, оставшиеся 2 байта это номер инструкции.

Операционный код регистра – 10h.

Например, внутреннее представление команды BRAN R1,R2,5 будет 71120005h.

***Разработка интерфейса эмулятора БРМ***

Основные функции разрабатываемой программы:

* трансляция исходного кода программы во внутреннее представление;
* возможность их просмотра
* ввод и просмотр значений регистров;
* возможность ввода исходного кода с текстового файла;
* выполнение исходного кода;
* пошаговая отладка.

Исходя из выше перечисленных функциональных требований, был разработан нижеприведенный интерфейс программы (рис. 2.1) и написаны основные функции, описанные в п. 4.1.

Интерфейс программы состоит из:

* компонента ввода исходного кода;
* компонента вывода транслированного кода;
* компонентов ввода и отображения регистров;
* кнопок трансляции исходного кода программы во внутреннее представление, пошаговой отладки и выполнения исходного кода;
* главного меню.

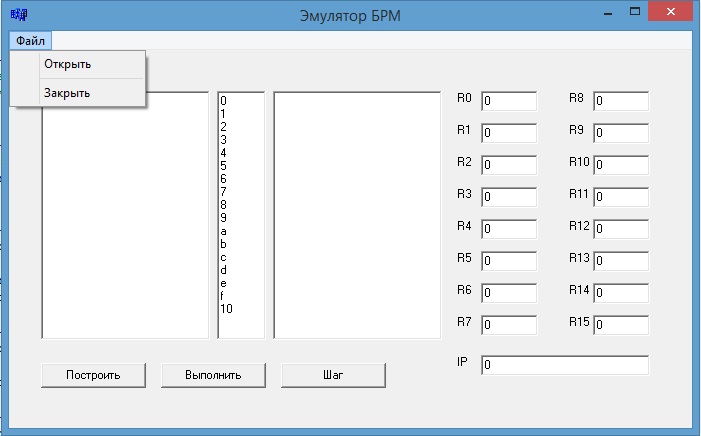


Рис. 1. Интерфейс эмулятора БРМ

**Описание программного эмулятора БРМ**

Программа "Эмулятор БРМ" предназначена для интерпретации выполнения программ регистровой машины путем трансляции кода программы во внутреннее представление. Программа написана на языке C++ и реализована в среде разработки Borland C++ Builder 6.0. Выбор обоснован широким распространением, а также удобством использования для разработки пользовательского интерфейса.

***Функциональное назначение***

Программа предназначена для выполнения и пошаговой отладки исходного кода. Программа имеет удобный и простой для понимания пользовательский интерфейс.

Пользователь имеет возможность:

* Вводить исходный код или осуществить ввод с помощью текстового файла.
* Вводить и менять значения регистров.
* Воспользоваться пошаговой отладкой программы.

***Описание логики программного эмулятора БРМ***

Программа состоит из двух модулей:

* Модуль главного окна (реализации интерфейса главного окна);
* Модуль регистровой модели;

Модуль главного окна является главным модулем. В этом модуле хранится большая часть функций и написаны все обработчики на события для компонентов.

Модуль регистровой модели исполняет роль «центрального обрабатывающего устройства» разработанной программы. В этом модуле содержатся функции:

* выполнения команд (очистка регистра, увеличение на 1 содержания регистра, переход по равенству двух регистров);
* занесения значения в указанный регистр;
* получения значения указанного регистра;
* получения значения счетчика команд IP;
* изменения значения счетчика команд IP;

Входными данными для программы является исходный код, написанный в соответствии с синтаксисом БРМ, который может быть введен вручную пользователем либо с текстового файла.

Выходными данными являются: результаты операций в виде значений регистров БРМ; транслированный во внутреннее представление, исходный код; сообщения о возникших ошибках.

**Разработка программного эмулятора БРМ**

Для глубокого понимания алгоритмов выполнения основных функций и самой программы необходимо знать содержания и назначения описанных ниже компонентов.

***Описание функций***

voidStep()

Результат: выполнение команд (очистка регистра, увеличение на 1 содержания регистра, переход по равенству двух регистров).

Структурная схема алгоритма выполнения функции показана на рис. 4.2

void analyze(char \*\*instr, int &k, int parent, int &size)

Входные данные: инструкция, весовые коэффициенты k и parent, размер.

Результат: анализ исходного кода.

Структурная схема алгоритма выполнения функции показана на рис. 4.3



Рис. 2. Структурная схема алгоритма функции Step( )

void analyzeme()

Результат: анализ исходного кода, получение (посредством analyze) и сохранение весового коэффициента для каждой строки

Структурная схема алгоритма выполнения функции показана на рис. 2 - 6



Рис. 3. Структурная схема алгоритма функции Analyze()



Рис. 4. Структурная схема алгоритма функции Analyzeme

void TranslateRegister(ucharopcode, constchar\* reg, ulong offset)

Входные данные: opcode, название регистра, смещение

Результат: трансляция в машинный код.

Структурная схема алгоритма выполнения функции показана на рис. 4.5

void Translator(char \*\*instr, int &k, ulong offset)

Входные данные: инструкция, весовой коэффициент, смещение.

Результат: транслирует операнды (посредством TranslateRegister) и команды.

Структурная схема алгоритма выполнения функции показана на рис. 4.6



Рис.5. Структурная схема алгоритма функции TranslateRegister

void printOpCodes()

Результат: вывод машинного кода в компонент вывода машинного кода.

void setR(int narg, int value)

Входные данные: номер регистра, значение.

Результат: присваивает значение регистру.

int getR(intnarg)

Входные данные: номер регистра.

Результат: получает значение регистра.

int getIP()

Результат: получает значение указателя.

voidsetCorrect()

Результат: корректировка IP.

void clear()

Результат: очистка компонентов модуля регистровой модели.



Рис. 6. Структурная схема алгоритма функции Translator

***Описания обработчиков компонентов типа «Button»***

На рис. 7 отображен пользовательский интерфейс эмулятора БРМ.

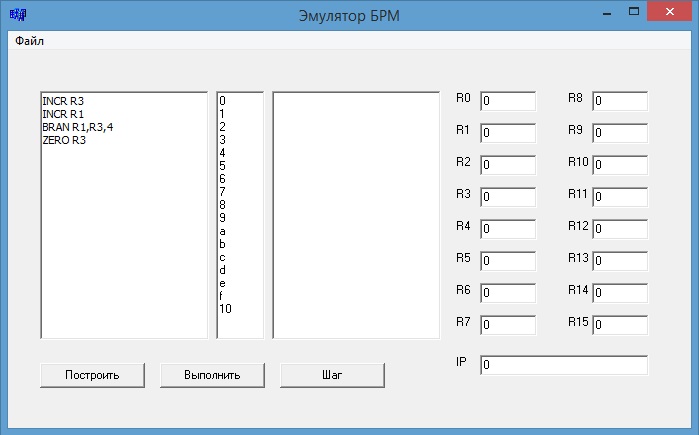


Рис.7. Пользовательский интерфейс

В этом обработчике выполняется транслирование исходного кода, написанного в компоненте для отображения исходного кода, в машинный код и его вывод в компонент отображения транслированного кода. Структурная схема алгоритма выполнения кода обработчика отображена на рис. 8.



Рис. 8. Структурная схема преобразования исходного кода в машинный

**Работа с программным эмулятором БРМ**

***Руководство пользователя***

Интерфейс программы прост и удобен для пользования.

На рис. 9 изображен интерфейс главного окна программы.

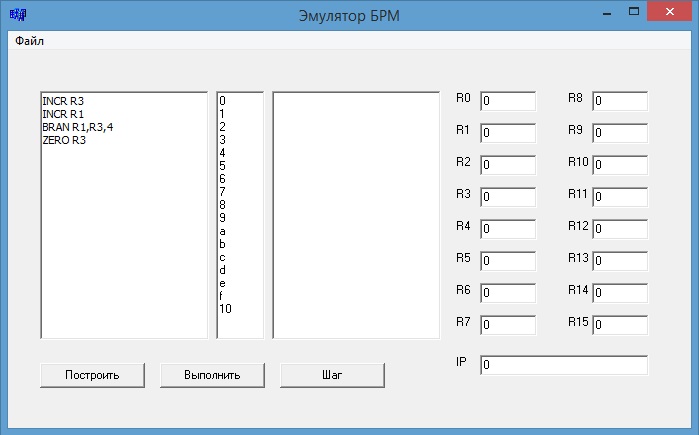


Рис.9. Интерфейс главного окна программы

Ввод исходного кода и значений регистров осуществляется нажатием мышью на соответствующее поле и вводом кода при помощи клавиатуры.

При нажатии мышкой по надписи «Файл» открывается главное меню, как показано на рис.10.

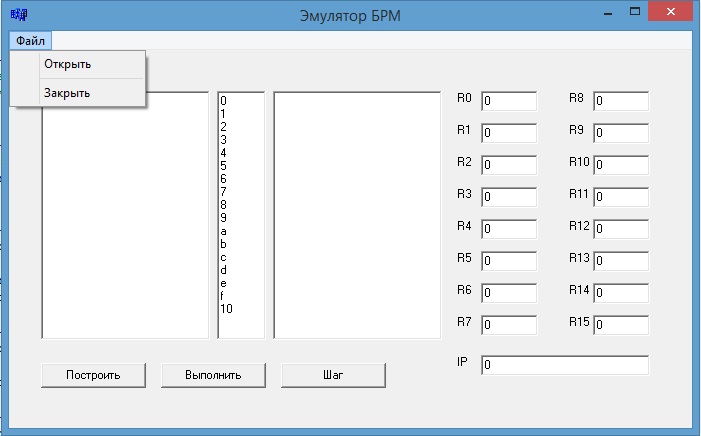


Рис 10. Основное меню главного окна

В состав главного меню входят 2 пункта:

1. При выборе пункта «Открыть» вызывается диалоговое окно, с помощью которого можно произвести ввод текста исходного кода в левый текстовый компонент с .txt файла. См. рис. 5.3.
2. При выборе пункта «Закрыть» выполняется закрытие программы.

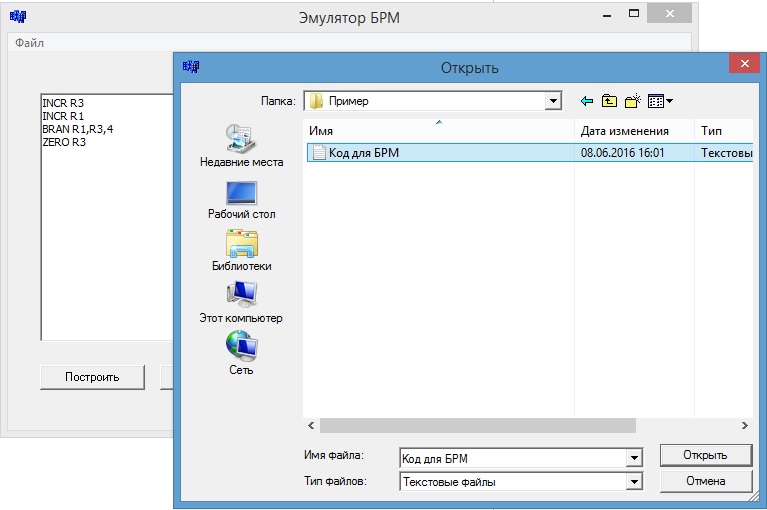


Рис. 11 Ввод с текстового файла

При нажатии кнопки «Построить» выполняется трансляция исходного кода, находящегося в левом текстовом компоненте, в машинный код и вывод на правый текстовый компонент. До трансляции в машинный код, выполнение исходного кода осуществляться не будет.

Важный момент: выполнение и отладка кода производится в соответствии с машинным кодом.

При нажатии на кнопку «Выполнить» производится выполнение транслированного кода программы и обновление значений регистров в соответствующих полях.

При нажатии на кнопку «Шаг» производится операция под указанным в регистре IP номером.

***Проверка работоспособности эмулятора БРМ с помощью тестов***

Программа была протестирована на примере исходных программ, в которых использовались различные команды. В тестах варьировались также значения регистров. При выполнении всех тестовых комбинаций поведение эмулятора БРМ, конечные и промежуточные значения регистров соответствовали результатам, полученным вручную. Далее приводится пример работы эмулятора с базовыми командами (см. рис. 12 и рис. 13).

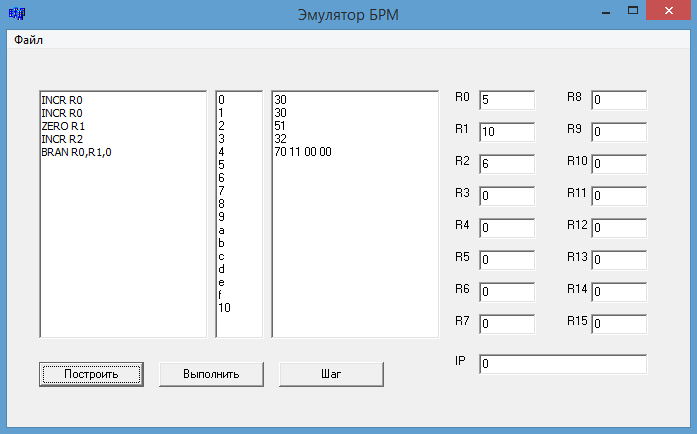


Рис. 12 Эмулятор БРМ после трансляции исходного кода во внутреннее представление

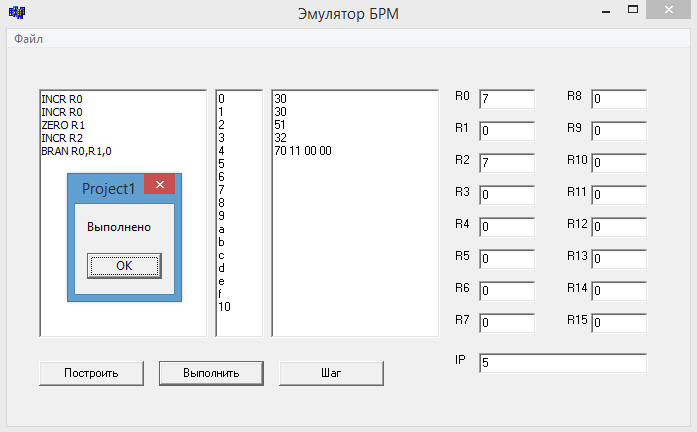


Рис. 13 Эмулятор БРМ после выполнения транслированного кода

**Заключение**

В результате выполненной работы разработан эмулятор базовой регистровой машины. Основой для разработки послужила базовая регистровая машина с ограниченным набором команд, структура которой определена как набор регистров общего назначения, хранящих натуральные числа, а алгоритм функционирования машины описан как процесс выполнения конструкций языка на заданной структуре.

В программе реализована возможность просмотра промежуточных и конечных результатов.

Программа имеет интерфейс, позволяющий вводить и изменять исходные данные, как с помощью непосредственного ввода, так и с помощью считывания из текстового файла.

Система реализована и прошла несколько этапов тестирования, что соответствует требованиям ТЗ.

**Литература**

1. Бьерн Страуструп, Язык программирования С++ , 2004. – 369 с. Джаррод 2. Холингворт, Боб Сворт, Марк Кэшмэн, Поль Густавсон. Borland C++ Builder 6. Руководство разработчика. 2004. — С. 976

3. ГОСТ 19.701-90 ЕСПД Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения.

4. Кочетков Ю.Ю. Вычислимые функции.

5. Сайт «Основы программирования на языках Си и С++» cppstudio.com