

СОДЕРЖАНИЕ

Введение	2
1 Технический проект	3
1.1 Анализ предметной области	3
1.1.1 Архитектура виртуального компьютера ММІХ	3
1.1.2 Память и регистры ММІХ	3
1.1.3 Команды ММІХ	4
1.2 Технология обработки информации.....	4
1.3 Входные и выходные данные.....	7
1.4 Системные требования	7
2 Рабочий проект	8
2.1 Общие сведения о работе системы.....	8
2.2 Функциональное назначение программного продукта	8
2.3 Инсталляция и выполнение программного продукта.....	8
2.4 Общий алгоритм программного продукта.....	8
2.5 Разработанные меню и интерфейсы.....	15
2.6 Сообщения системы.....	15
3 Программа и методика испытаний	16
Заключение.....	17
Список использованных источников.....	18
Приложение Тестовые программы	19

ВВЕДЕНИЕ

Виртуальная машина — это абстрактная машина, создаваемая для облегчения переносимости программ. Для этого виртуальная машина реализует ту часть работы, которая зависит от реализации конкретной машины. Некоторые виртуальные машины эмулируют работу аппаратного обеспечения. Для этого они воссоздают работу процессора, памяти и содержат систему команд.

Цель: разработать виртуальную машину, описанную в книге [1].

Назначение: изучить архитектуру виртуальных машин системы RISC и особенности их разработки.

1 ТЕХНИЧЕСКИЙ ПРОЕКТ

1.1 Анализ предметной области

1.1.1 Архитектура виртуального компьютера MMIX

Виртуальная машина — программная и/или аппаратная система, эмулирующая аппаратное обеспечение некоторой платформы (target — целевая, или гостевая платформа) и исполняющая программы для target-платформы на host-платформе (host — хост-платформа, платформа-хозяин) или виртуализирующая некоторую платформу и создающая на ней среды, изолирующие друг от друга программы и даже операционные системы, также спецификация некоторой вычислительной среды (например: «виртуальная машина языка программирования Си»).

RISC — архитектура процессора, в котором быстродействие увеличивается за счёт упрощения инструкций, чтобы их декодирование было более простым, а время выполнения — меньшим.

MMIX является 64-битовым RISC-компьютером, содержащим 256 регистров общего назначения и 64-битовое адресное пространство.

Интерпретатор — это программная модель исполнителя команд абстрактной (виртуальной) машины.

Загрузчик — это программа, выполняющая загрузку массива слов (программ и данных) с внешнего устройства в память компьютера.

Программа для виртуальной машины (вместе с данными) хранится в файле и перед интерпретацией считывается в память. Затем интерпретатор начинает выполнять команды, выбирая их из памяти.

Загружаемый в память массив слов может содержать команды, вещественные числа, целые числа и должен заканчиваться маркером конца.

Массив слов представляет собой текстовый файл, его нужно набрать в текстовом редакторе (например, в Блокноте).

1.1.2 Память и регистры MMIX

В компьютере MMIX имеется 2^{64} байт памяти, 256 регистров общего назначения и 32 специальных регистра. Данные передаются из памяти в регистры, преобразуются в регистрах и передаются из регистров в память. Ячейки памяти обозначаются $M[0], M[1], \dots, M[2^{64} - 1]$. Регистры общего назначения обозначаются $\$0, \$1, \dots, \$255$.

2^{64} байт памяти сгруппированы в 2^{63} вайда: $M_2[0] = M_2[1] = M[0]M[1]$, $M_2[2] = M_2[3] = M[2]M[3]$, ...; каждый вайд состоит из двух последовательных байтов $M[2k]M[2k+1] = M[2k] * 2^8 + M[2k+1]$ и обозначается $M_2[2k]$ или $M_2[2k + 1]$. Аналогично, 2^{64} байт памяти сгруппированы в 2^{62} тетрабайта $M_4[4k] = \dots = M_4[4k + 3] = M[4k]M[4k + 1] \dots M[4k + 3]$, и 2^{61} октабайта $M_8[8k] = \dots = M_8[8k + 7] = M[8k]M[8k + 1] \dots M[8k + 7]$. Если x – октабайт, то

$M_2[x]$, $M_4[x]$, $M_8[x]$ обозначают вайд, тетра и окта, которые содержат байт $M[x]$. Компьютер MMIX поддерживает целые числа (4 и 8 байтов) и дробные (4 и 8 байтов) числа с плавающей точкой.

1.1.3 Команды MMIX

В памяти MMIX хранятся не только данные, но и инструкции. Инструкция, или «команда», – это тетрабайт, состоящий из 4 байтов: OP, X, Y и Z. Байт OP содержит код операции (или коротко «опкод»); а байты X, Y, Z – операнды (команды записываются в шестнадцатеричной системе счисления). Данные в регистрах могут быть целыми или дробными числами с плавающей точкой.

В таблице 1.1 приведен список кодов операций, используемых процессором.

Таблица 1.1 – Список кодов операций

Тип	Коды
Операции загрузки и сохранения	0x80 – 0x93, 0xA0 – 0xB5
Арифметические операции	0x18 – 0x33
Операции условного перехода	0x60 – 0x7F
Операции с плавающей точкой	0x01 – 0x17
Операции безусловного перехода и ветвления	0x40 – 0x4F, 0xF0 – 0xF1, 0x9E – 0x9F
Команда окончания работы программы (STOP)	0xF9

1.2 Технология обработки информации

На диаграмме классов (рис. 1.1) представлены элементы виртуальной машины. Процессора представлен классом Processor, команды представлены классом Command. От класса Command наследуются все команды виртуальной машины. В классе CPU хранится массив указателей на объекты команд.

На рис. 1.1 представлена диаграмма классов.

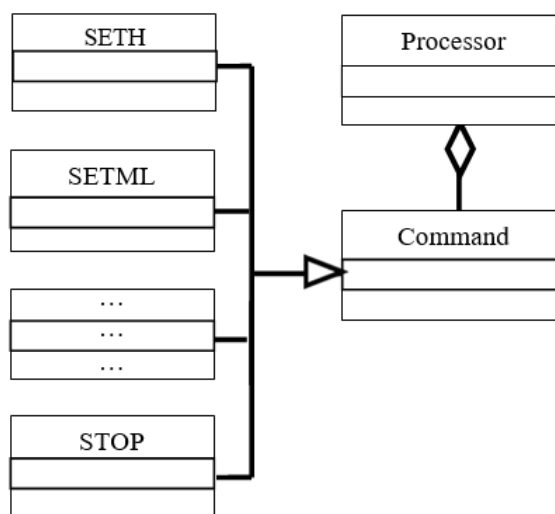


Рисунок 1.1 – Диаграмма классов

Программа рассчитана на одного пользователя, который должен подготовить файл с кодом исполняемой программы.

Формат файла программы: в начале каждой строки должен стоять тег (таб. 1.2), после которого через пробел идет число в десятичной системе счисления (если тег 'a'), команда в шестнадцатеричном виде (если тег 'c'), адрес и число в десятичном виде (если тег 'i' или 'f'). После каждой команды в строке через пробел может стоять комментарий.

Таблица 1.2 – Список тегов файла программы

Тег	Назначение
a	Адрес загрузки в память и стартовый адрес программы (в десятичной системе счисления)
c	Команда (8 байтов в шестнадцатеричном виде без пробелов)
i	Целое число (в десятичной системе счисления)
f	Число с плавающей точкой

Алгоритм загрузки программы в память

Начало алгоритма

```

|   Индексу элемента памяти процессора index присвоить 0;
|   Если файл программы найден, то
|   |   Пока не конец файла
|   |   |   Считать строку из файла;
|   |   |   Если равен 'a', то
|   |   |   |   Указателю начала программы IP присвоить второй элемент строки;
|   |   |   |   index присвоить IP;
|   |   |   А если первый элемент строки равен 'c', то
|   |   |   |   Второй элемент строки разбить на 4 части по 2 символа и записать в
|   |   |   |   |   память процессора, начиная с index;
|   |   |   |   Увеличить index на 4;
|   |   |   А если равен 'i', то
|   |   |   |   Если третий элемент строки больше -129 и меньше 128, то
|   |   |   |   |   Записать это число в память процессора как байт по адресу, равному
|   |   |   |   |   |   второму элементу данной строки;
|   |   |   |   А если третий элемент строки больше -32769 и меньше 32769, то
|   |   |   |   |   Записать это число в память процессора как вайд по адресу, равному
|   |   |   |   |   |   второму элементу данной строки;
|   |   |   А если третий элемент строки больше -2147483649 и меньше
|   |   |   |   2147483648, то

```

```

|   |   |   |   |   Записать это число в память процессора как тетрабайт по адресу,
                        равному второму элементу данной строки;
|   |   |   |   |   Иначе
|   |   |   |   |   Записать это число в память процессора как октабайт по адресу,
                        равному второму элементу данной строки;
|   |   |   |   |   Конец ветвления
|   |   |   |   |   А если равен 'f', то
|   |   |   |   |   Третий элемент строки записать в память процессора как октабайт по
                        адресу, равному второму элементу данной строки;
|   |   |   |   |   Конец ветвления
|   |   |   |   |   Конец цикла

```

Завершение алгоритма

Алгоритм работы виртуального процессора

memory – массив памяти процессора;

IP – индекс операции в массиве memory;

Начало алгоритма

```

|   Пока memory[IP] не равен коду команды STOP
|   |   Выполнить команду из массива команд процессора, по коду операции
        memory[IP];
|   |   Если не был совершен условный или безусловный переход, то
|   |   |   Увеличить IP на 4;
|   |   Конец цикла

```

Завершение алгоритма

Алгоритм работы программы

Начало алгоритма

```

|   Запросить у пользователя имя файла;
|   Если файл не найден, то
|   |   Вывести сообщение об ошибке;
|   |   Иначе
|   |   |   Выполнить алгоритм загрузки программы в память;
|   |   |   Выполнить алгоритм интерпретатора;
|   |   |   Вывести на экран содержимое регистра $0;
|   |   Конец ветвления

```

Завершение алгоритма

Алгоритмы команд

IP – указатель на текущую команду.

@ – знак операции.

1. Арифметические операции с регистрами: $\$X = \$Y @ \$Z$.
2. Арифметические операции с регистром и числом: $\$X = \$Y @ Z$.
3. Сравнение значений регистров: $\$X = (\$Y > \$Z) - (\$Y < \$Z)$.

4. Операции условного перехода:

Если $\$X$ удовлетворяет условию, то $IP = YZ$.

5. Операции безусловного перехода

$\$X = IP + 4$;

$IP = \$Y + \Z .

1.3 Входные и выходные данные

Входные данные: файл программы.

Выходные данные: значения регистров и содержимое памяти процессора.

1.4 Системные требования

Рекомендуемая конфигурация:

- Intel-совместимый процессор с частотой не менее 1,6 ГГц;
- не менее 512 МБ ОЗУ;
- не менее 1 МБ свободного места на диске;
- дисковод CD-ROM/DVD-ROM;
- операционная система: Windows XP (x86) Professional (SP3) и более поздние;

2 РАБОЧИЙ ПРОЕКТ

2.1 Общие сведения о работе системы

Программный продукт разработан в среде Code::Blocks 13.12 на языке C++ (стандарт C++11). Программа работает под управлением операционной системы Windows XP (x86) Professional (SP3) и более поздними.

2.2 Функциональное назначение программного продукта

Разработанный программный продукт предназначен эмуляции работы процессора MMIX. Программа имеет следующие функциональные ограничения:

- максимальные и минимальные числа ограничены максимумом и минимумом типов long long и double;
- объем памяти процессора – 64 КБ, следовательно, адрес байта в памяти не может превышать 65536;
- Количество реализованных команд – 111.

2.3 Установка и выполнение программного продукта

Для выполнения программы необходимо:

1. Скопировать на жесткий диск компьютера папку mmix, содержащую проект и файлы тестовых программ («example1.txt», «example2.txt», «example3.txt» – см. приложение «Тестовые программы»).
2. Запустить файл mmix.exe, расположенный в папке mmix.

2.4 Общий алгоритм программного продукта

Программный продукт состоит из 5 модулей: «main.cpp», «Processor.h», «Processor.cpp», «Command.h» и «Command.cpp». Для реализации работы процессора был разработан класс Processor, для реализации команд процессора – класс Command.

Для работы с памятью и регистрами процессора были разработаны структуры данных типа union – Tetrabyte и Octabyte.

В таблице 2.1 приведены поля структуры Tetrabyte.

Таблица 2.1 – Поля структуры Tetrabyte

Имя	Тип	Назначение
tetra[4]	unsigned char	Представление тетрабайта в виде массива из 4 байтов
int32	int	Представление тетрабайта в виде целого знакового числа
uint32	unsigned int	Представление тетрабайта в виде целого беззнакового числа
fl	float	Представление тетрабайта в виде короткого числа с плавающей точкой

В таблице 2.2 приведены поля структуры Octabyte.

Таблица 2.2 – Поля структуры Octabyte

Имя	Тип	Назначение
octa[8]	unsigned char	Представление октабайта в виде массива из 8 байтов
int64	long long	Представление октабайта в виде целого знакового числа
uint64	unsigned long long	Представление октабайта в виде целого беззнакового числа
db	double	Представление октабайта в виде числа с плавающей точкой

В таблице 2.3 приведены методы класса Processor, используемого в программе.

Таблица 2.3 – Методы класса Processor

Прототип	Назначение
public: void interpretator()	Выполнение программы
public: void reset()	Очистка регистров и памяти

В таблице 2.4 приведены поля класса Processor.

Таблица 2.4 – Поля класса Processor

Имя	Тип	Назначение
memory	unsigned char*	Указатель на память
IP	unsigned long long*	Индекс текущей инструкции в памяти
commonRegisters[256]	Octabyte	Регистры общего назначения
specialRegisters[32]	Octabyte	Регистры специального назначения
commands[256]	Command*	Массив команд

Класс Command – это абстрактный класс, не имеющий полей и содержащий один метод – void operator()(Processor& cpu).

В таблице 2.5 приведен список классов-наследников класса Command и их назначение.

Таблица 2.5 – Список классов-наследников класса Command

Название	Назначение
class SETH	Установить старший вайд в регистр
class SETMH	Установить средний старший вайд в регистр
class SETML	Установить средний младший вайд в регистр
class SETL	Установить младший вайд в регистр
class INCH	Увеличить старший вайд в регистре
class INCMH	Увеличить средний старший вайд в регистре
class INCML	Увеличить средний младший вайд в регистре
class INCL	Увеличить младший вайд в регистре
class LDB	Загрузить байт из памяти в регистр
class LDW	Загрузить вайд из памяти в регистр
class LDT	Загрузить тетрабайт из памяти в регистр
class LDO	Загрузить октабайт из памяти в регистр
class LDBI	Загрузить байт из памяти в регистр
class LDWI	Загрузить вайд из памяти в регистр
class LDTI	Загрузить тетрабайт из памяти в регистр
class LDOI	Загрузить октабайт из памяти в регистр
class LDHT	Загрузить тетрабайт из памяти в старшую половину регистра
class LDHTI	Загрузить тетрабайт из памяти в старшую половину регистра
class STB	Сохранить байт из регистра в память
class STW	Сохранить вайд из регистра в память
class STT	Сохранить тетрабайт из регистра в память
class STO	Сохранить октабайт из регистра в память
class STBI	Сохранить байт из регистра в память
class STWI	Сохранить вайд из регистра в память
class STTI	Сохранить тетрабайт из регистра в память
class STOI	Сохранить октабайт из регистра в память
class STHT	Сохранить старшую половину регистра в память
class STHTI	Сохранить старшую половину регистра в память
class STCO	Сохранить константу в октабайте памяти
class STCOI	Сохранить константу в октабайте памяти
class ADD	Целочисленное сложение двух регистров

Продолжение таблицы 2.5

Название	Назначение
class ADDI	Целочисленное сложение регистра с константой
class ADDU	Целочисленное беззнаковое сложение двух регистров
class ADDUI	Целочисленное беззнаковое сложение регистра с константой
class SUB	Целочисленная разность двух регистров
class SUBI	Целочисленная разность регистра и константы
class SUBU	Целочисленная беззнаковая разность двух регистров
class SUBUI	Целочисленная беззнаковая разность регистра и константы
class MUL	Целочисленное произведение двух регистров
class MULI	Целочисленное произведение регистра и константы
class DIV	Целочисленное деление двух регистров
class DIVI	Целочисленное деление регистра и константы
class NEG	Отрицание целочисленного числа в регистре
class NEGI	Отрицание целочисленного беззнакового числа в регистре
class CMP	Целочисленное сравнение двух регистров
class CMPI	Целочисленное сравнение регистра и константы
class CSN	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда отрицательное
class CSNI	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда отрицательное
class CSZ	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда равно нулю
class CSZI	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда равно нулю
class CSP	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда положительное
class CSPI	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда положительное
class CSOD	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда нечетное
class CSODI	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда нечетное

Продолжение таблицы 2.5

Название	Назначение
class CSEV	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда четное
class CSEVI	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда четное
class CSNN	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда неотрицательное
class CSNNI	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда неотрицательное
class CSNZ	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда не равно нулю
class CSNZI	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда не равно нулю
class CSNP	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда неположительное
class CSNPI	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда неположительное
class ZSN	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда отрицательное, иначе присвоить 0
class ZSNI	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда отрицательное, иначе присвоить 0
class ZSZ	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда равно нулю, иначе присвоить 0
class ZSZI	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда равно нулю, иначе присвоить 0
class ZSP	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда положительное, иначе присвоить 0

Продолжение таблицы 2.5

Название	Назначение
class ZSPI	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда положительное, иначе присвоить 0
class ZSOD	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда нечетное, иначе присвоить 0
class ZSODI	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда нечетное, иначе присвоить 0
class ZSEV	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда четное, иначе присвоить 0
class ZSEVI	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда четное, иначе присвоить 0
class ZSNN	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда неотрицательное, иначе присвоить 0
class ZSNNI	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда неотрицательное, иначе присвоить 0
class ZSNZ	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда не равно нулю, иначе присвоить 0
class ZSNZI	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда не равно нулю, иначе присвоить 0
class ZSNP	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда положительное, иначе присвоить 0
class ZSNPI	Условное присвоение первому операнду значение третьего операнда, если значение отрицательное 2 операнда положительное, иначе присвоить 0
class FADD	Сложение чисел с плавающей точкой
class FSUB	Вычитание чисел с плавающей точкой
class FMUL	Умножение чисел с плавающей точкой

Продолжение таблицы 2.5

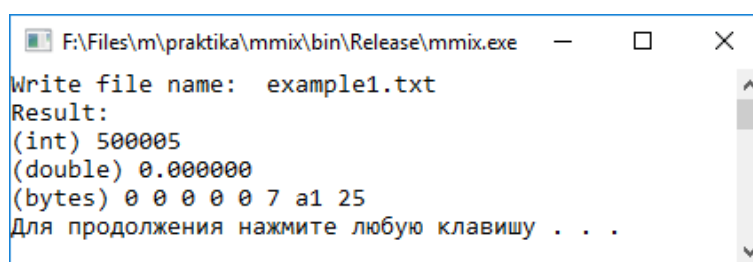
Название	Назначение
class FDIV	Деление чисел с плавающей точкой
class FSQRT	Извлечение корня из числа с плавающей точкой
class FINT	Получение целой части числа с плавающей точкой
class FCMP	Сравнение двух чисел с плавающей точкой
class FIX	Преобразование числа с плавающей точкой к целому типу
class FLOT	Преобразование к числу с плавающей точкой
class FLOTI	Преобразование к числу с плавающей точкой
class JMP	Перейти по относительному адресу вперед
class JMPB	Перейти по относительному адресу назад
class GO	Перейти по абсолютному адресу
class GOI	Перейти по абсолютному адресу
class BN	Условный переход по относительному адресу вперед, если значение первого операнда отрицательное
class BNB	Условный переход по относительному адресу назад, если значение первого операнда отрицательное
class BZ	Условный переход по относительному адресу вперед, если значение первого операнда равно нулю
class BZB	Условный переход по относительному адресу назад, если значение первого операнда равно нулю
class BP	Условный переход по относительному адресу вперед, если значение первого операнда положительное
class BPB	Условный переход по относительному адресу назад, если значение первого операнда положительное
class BOD	Условный переход по относительному адресу вперед, если значение первого операнда нечетное
class BODB	Условный переход по относительному адресу назад, если значение первого операнда нечетное
class BNN	Условный переход по относительному адресу вперед, если значение первого операнда неотрицательное
class BNNB	Условный переход по относительному адресу назад, если значение первого операнда неотрицательное

Продолжение таблицы 2.5

Название	Назначение
class BNZ	Условный переход по относительному адресу вперед, если значение первого операнда не равно нулю
class BNZB	Условный переход по относительному адресу назад, если значение первого операнда не равно нулю
class BNP	Условный переход по относительному адресу вперед, если значение первого операнда неположительное
class BNPB	Условный переход по относительному адресу назад, если значение первого операнда неположительное
class BEV	Условный переход по относительному адресу вперед, если значение первого операнда четное
class BEVB	Условный переход по относительному адресу назад, если значение первого операнда четное
class GET	Присвоить регистру общего назначения значение специального регистра
class PUT	Присвоить специальному регистру значение регистра общего назначения
class STOP	Завершение работы программы

2.5 Разработанные меню и интерфейсы

После запуска программы на экране сообщение с предложением ввести имя файла с программой, после чего предоставляется результат работы программы (рис. 2.1).



```

F:\Files\m\praktika\mmix\bin\Release\mmix.exe
Write file name: example1.txt
Result:
(int) 500005
(double) 0.000000
(bytes) 0 0 0 0 7 a1 25
Для продолжения нажмите любую клавишу . . .

```

Рисунок 2.1 – Консоль программы

2.6 Сообщения системы

В таблице 2.6 приведены сообщения системы.

Таблица 2.6 – Сообщения системы

№ п/п	Сообщение	Причина возникновения, способ устранения
1	File doesn't exist	Файл программы с заданным именем не существует
2	Result	Программа завершилась

3 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

1. Запустить программу на выполнение. Появится окно с предложением ввести имя файла с программой (см. рис. 2.1).
2. Ввести имя файла, которого нет в папке с программой, убедиться, что получено сообщение 1 (см. табл. 2.6).
3. Запустить программу на выполнение. Ввести имя файла «example1.txt», убедиться, что получено сообщение 2 и результат равен 500005 (см. табл. 2.6).
4. Запустить программу на выполнение. Ввести имя файла «example2.txt», убедиться, что получено сообщение 2 и результат равен 15.5 (см. табл. 2.6).
5. Запустить программу на выполнение. Ввести имя файла «example3.txt», убедиться, что получено сообщение 2 и результат равен 120 (см. табл. 2.6).

ЗАКЛЮЧЕНИЕ

В результате курсового проектирования была разработана виртуальная машина процессора MMIX.

Программа отвечает поставленным требованиям и может быть использована для эмулирования работы процессора MMIX.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кнут Д. Искусство программирования, том 1, выпуск 1. MMIX – RISC-компьютер нового тысячелетия – М.: ООО «И.Д. Вильямс», 2007. – 160с
2. Лаптев В.В. – С++. Объектно-ориентированное программирование: Учебное пособие. – СПб.: Питер, 2008. – 464 с.
3. Готтшлинг П. Современный С++ для программистов, инженеров и ученых. – М.: ООО «И.Д. Вильямс», 2016. - 512 с.

Тестовые программы

1) Программа «example1», вычисляющая сумму чисел 500000 и 5.

```
a 16          // устанавливаем начало программы в памяти
с E301C350    // устанавливаем младший вайд в регистре $1
i 50000 500000 // сохраняем число 500000 в память по адресу 50000
с 89010100    // загружаем тетрабайт в регистр $1
с 21000105    // $0 = $1 + 5
с F9000000    // конец программы
```

2) Программа «example2», вычисляющая произведение чисел 10 и 1.55.

```
a 16          // устанавливаем начало программы в памяти
с E3022722    // устанавливаем младший вайд в регистре $2
f 10018 10    // сохраняем число 10 в память по адресу 10018
с 8D020200    // загружаем октабайт из памяти в регистр $2
с E3034E32    // устанавливаем младший вайд в регистре $3
f 20018 1.55  // сохраняем число 1.55 в память по адресу 20018
с 8D030300    // загружаем октабайт из памяти в регистр $3
с 10000203    // умножаем регистры $2 и $3
с F9000000    // конец программы
```

3) Программа «example3», вычисляющая факториал числа 5.

```
a 400          // устанавливаем начало программы в памяти
с E3000190    // $0 = 400 (начало программы)
с F000001C    // -----тут начинается подпрограмма -----
с E3020001    // $2 = 1
с 21030301    // $3 += 1
с 18020203    // $2 *= $3
с 30040301    // $4 = ($3 > $1) - ($3 < $1), т.е. $4 будет = -1, если $3 < $1
с 4104000C    // BNB - переход на 12 байт вверх (на 3 команды)
с 9F050500    // возвращаемся туда, откуда пришли -----
с E3010005    // $1 = 5 (функция возвратит 5! = 120)
с 9F050008    // вызываем подпрограмму и сохраняем в $5 IP для возврата
с 21000200    // $0 = $2 + 0 (записываем в $0 результат)
с F9000000    // конец программы
```