Санкт-Петербургский государственный политехнический университет
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчёт по курсовой работе

Дисциплина: Высокоуровневое моделирование средствами SystemC

Тема: Разработка потактовой модели процессора

Выполнил студент гр. 13541/2 _____ В.Е. Бушин
(подпись)
Руководитель _____ О.В. Мамутова
(подпись)

“_” _____2017 г.

Санкт-Петербург
2017

Программа работы:
Раздел 1. Спецификация компонентов процессорной системы.
Обязательные компоненты системы: АЛУ, контроллер памяти, регистровый файл.
Дополнительные компоненты системы по индивидуальному заданию:
- контроллер прямого доступа к памяти (DMA)
- поддержка виртуальной памяти (MMU)
- интерфейс ввода/вывода (GPIO)
Предусмотреть отладочный режим работы модулей.

Раздел 2. Спецификация системного окружения для отладки и тестирования процессора.
Допущения:
Время доступа к внешней памяти — от 7 процессорных тактов.

Раздел 3. Спецификация системы команд процессора типа RISC, исходя из выбранного набора компонентов процессорной системы и целевых алгоритмов.

Раздел 4. Разработка описания ядра процессора, компонентов процессорной системы и элементов системного окружения на языке SystemC.

Раздел 5. Модульное тестирование разработанной системы.

Раздел 6. Создание программы в машинных кодах для реализации заданных алгоритмов. Демонстрация работоспособности процессора. Оценка эффективности выполнения алгоритмов.

Выполнение работы

Функциональная схема процессорной системы приведена на рис. 1.

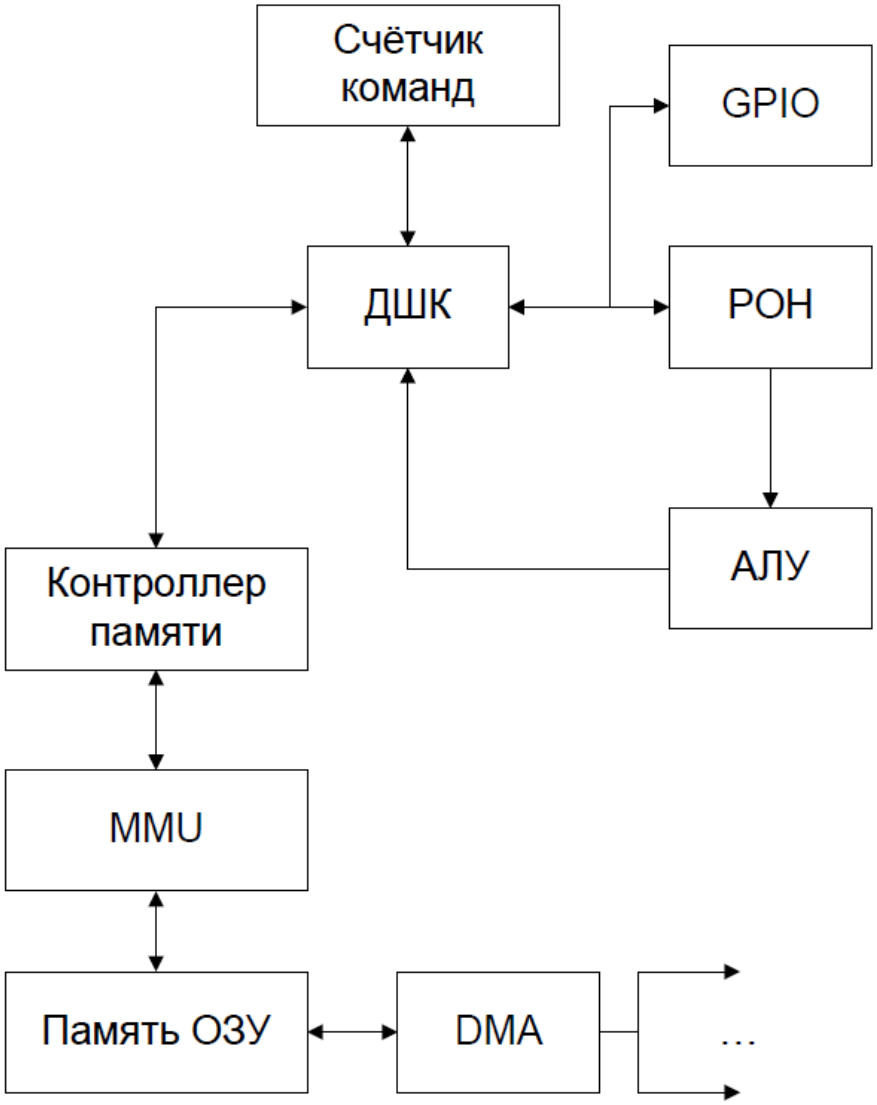

Рис.1. Функциональная схема процессорной системы.

Процессор состоит из дешифратора команда (ДШК), АЛУ, регистрового файла (РОН), счётчика команд, контроллера памяти, MMU, DMA. Управление работой процессора осуществляет дешифратор команд: сначала дешифратор подаёт сигнал контроллеру памяти на чтение команды из памяти по адресу, указанному в счётчике команд, затем после получения команды дешифратор определяет код операции и генерирует управляющие сигналы на другие модули, после выполнения команды дешифратор подаёт сигнал на чтение новой команды из памяти.

Действия выполняемые АЛУ:

Код операции	Операция
1	Сложение
2	Вычитание
3	Инкремент
4	Декремент

5	Умножение
6	Сдвиг влево
7	Сдвиг вправо

АЛУ может формировать флаги нуля и переноса. Все операнды берутся из РОН.

Система команд

Команды (кроме АЛУ):

- Загрузка из памяти в регистр (косвенная/прямая);
- Сохранение из регистра в память (косвенное/прямое);
- Пересылка из регистра в регистр;
- Загрузка регистра непосредственная;
- Безусловный переход;
- Переход по нулевому/ненулевому признаку;
- Переход при наличии/отсутствии переноса;
- Вызов подпрограммы/возврат из подпрограммы.

Коды всех команд хранятся в дешифраторе следующим образом:

```
enum instructionCode{
              //команды для АЛУ
              ADD = 1, //! сложение
              SUB = 2, //! вычитание
              INR = 3, //! инкремент
              DCR = 4, //! декремент
              MULT = 5, //! умножение
              SHIFT_LEFT = 6, //! сдвиг влево
             SHIFT_RIGHT = 7, //! сдвиг вправо

              //команды переходов
              JMP = 0x10, //! безусловный переход
              JNZ = 0x11, //! переход по ненулевому результату
              JZ = 0x12, //! переход по нулевому результату
              JNC = 0x13, //! переход при отсутствии переноса
              JC = 0x14, //! переход при наличии переноса
              CALL = 0x15, //! вызов подпрограммы
              RET = 0x16, //! возврат из подпрограммы

              //команды для работы с памятью
              LOAD_I = 0x21, //! загрузка регистра непосредственная
              MOV = 0x22, //! пересылка из регистра в регистр
              LOAD_M = 0x23, //! загрузка из памяти в регистр
              MOV_M = 0x24, //! сохранение из регистра в память
              LOAD_MI = 0x25, //! косвенная загрузка из памяти в регистр
              MOV_MI = 0x26, //! косвенное сохранение из регистра в память

              //команды для работы с GPIO
              LOAD_IO = 0x27, //! считывание из блока GPIO
              MOV_IO = 0x28 //! запись в блок GPIO
       };
```

Команды для АЛУ с 2-мя операндами (сложение, вычитание, умножение), операнды берутся из РОН и результат сохраняется в РОН:

Код операции	Не использ.	Операнд A	Операнд B	Адрес результата
6 бит	11 бит	5 бит	5 бит	5 бит

Команды для АЛУ с одним операндом (инкремент, декремент, сдвиг влево, сдвиг вправо), операнд берётся из РОН и результат сохраняется в РОН:

Код операции	Не использ.	Операнд	Адрес результата
6 бит	16 бит	5 бит	5 бит

Загрузка регистра непосредственная:

Код операции	Адрес регистра	Данные
6 бит	5 бит	21 бит

Пересылка из регистра A в регистр B:

Код операции	Не использ.	Регистр A	РегистрB
6 бит	16 бит	5 бит	5 бит

Прямая загрузка из памяти в регистр / прямое сохранение из регистра в память:

Код операции	Не использ.	Адрес в памяти	Регистр РОН
6 бит	11 бит	10 бит	5 бит

Косвенная загрузка из памяти в регистр / косвенное сохранение из регистра в память:

Код операции	Не использ.	Регистр адреса	Регистр данных
6 бит	16 бит	5 бит	5 бит

Чтение из GPIO:

Код операции	Не использ.	Регистр РОН
6 бит	21 бит	5 бит

Запись в GPIO:

Код операции	Не использ.	Данные
6 бит	18 бит	8 бит

Команды переходов (безусловный переход, переход по нулевому/ненулевому результату, при наличии/отсутствии переноса, вызов подпрограммы):

Код операции	Не использ.	Адрес
6 бит	16 бит	10 бит

Команда возврата из подпрограммы:

Код операции	Не использ.
0x16	26 бит

DMA

Блок DMA реализован в виде конечного автомата. Данный блок предоставляет доступ внешним устройствам к запрашиваемой ячейке памяти на запись или чтение. Одновременно обрабатывается только один запрос на запись или чтение от одного устройство, другое запрашивающие устройство будет ожидать завершения обработки блоком DMA первого устройства.

Проверка блока DMA приведена на рис. 2.

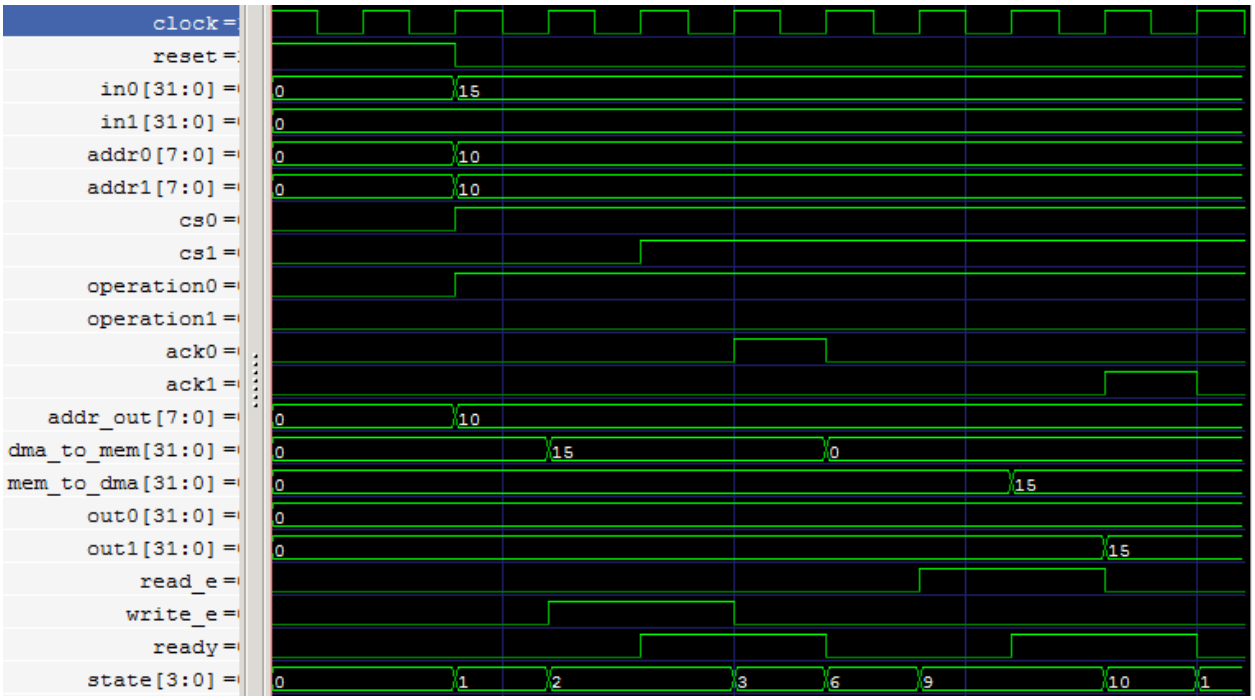

Рис.2. Проверка блока DMA.

На данном тесте сначала 0 устройство запрашивает DMA на запись в ячейку по адресу 10 числа 15, затем через 2 такта 1 устройство запрашивает чтение из ячейки 10. Но в этот момент ещё идёт обработка 0 устройства, поэтому 1 устройство пока ожидает своей очереди. После записи числа 15 в ячейку 10 DMA выдаёт сигнал готовности 0 устройству (ack0) и переходит к обработке запроса от другого устройства. В итоге устройство 1 получает данные из ячейки памяти 10 (out1) и сигнал готовности (ack1).

GPIO

Блок GPIO имитирует работу с внешним периферийным устройством. Блок имеет вход и выход на периферийное устройство, размерность данных – 8 бит. На рис. 3 приведён тест GPIO.

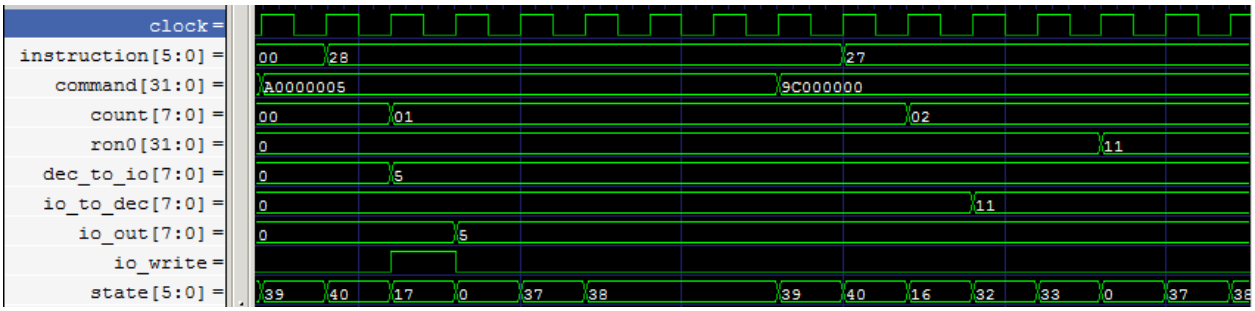

Рис. 3. Тест блока GPIO.

На данном тесте проверяются команды записи в блок GPIO и чтения из блока GPIO. Сначала идёт команда записи в блок GPIO: записывается число 5 – и затем на выходе блока GPIO мы видим это число (io_out). После команды записи идёт команда чтения, на вход блока GPIO подаётся число 11 и в итоге данное число записывается в РОН 0, что говорит о корректности работы данного блока.

MMU

Блок MMU реализует трансляцию адресов виртуальной памяти в адреса физической памяти, а также загрузку необходимых страниц памяти из «жёсткого диска» в ОЗУ. На рис. 4 приведено содержимое ОЗУ после теста.

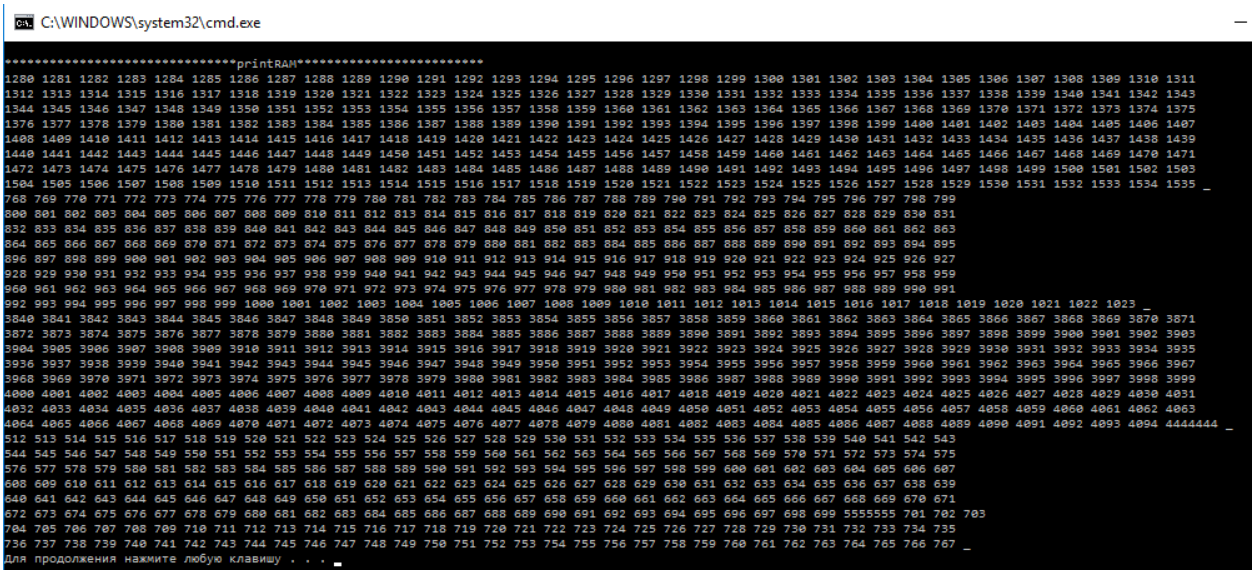

Рис. 4. Содержимое ОЗУ после теста.

Данный тест проверяет корректность выгрузки страниц с «жёсткого диска» в ОЗУ и преобразования виртуального адреса в физический. Страница имеет

размер 256 бит. Т.е. в ОЗУ одновременно может храниться 4 страницы. По запросу от контроллера памяти блок MMU загружает страницу с требуемыми данными в ОЗУ. Затем уже читает из этой страницы необходимую ячейку памяти и передаёт её контроллеру памяти. Блок MMU по мере необходимости подгружает страницы в ОЗУ и выгружает старые страницы из ОЗУ на «жёсткий диск». По виртуальному адресу 4095 было записано число 4444444, а по виртуальному адресу 700 число 5555555. По тесту видно, что данные числа были корректно записаны в ОЗУ и отображаются на правильных позициях. Также при выводе содержимого «жёсткого диска» записанные в ОЗУ данные были успешно переписаны в него (рис.5), что говорит о корректной работе блока MMU.

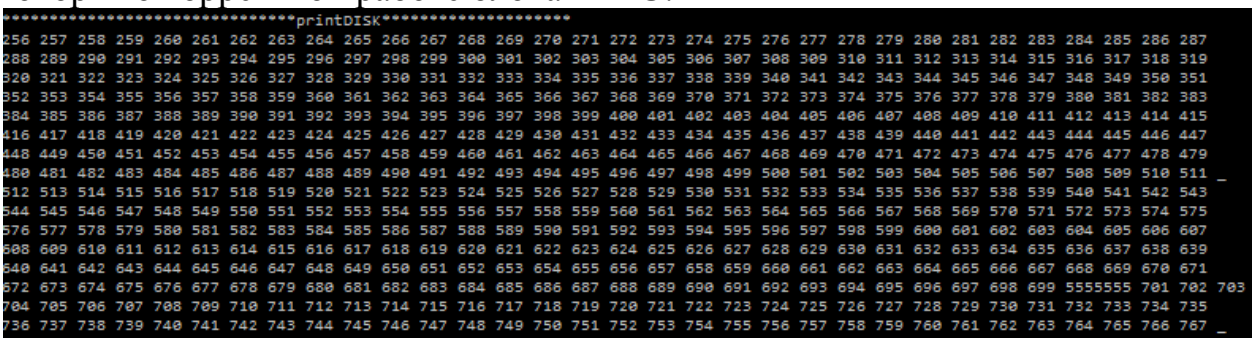

Рис. 5. Содержимое «жёсткого диска» после теста.

Алгоритмы

1. **Сортировка для массива из 16 слов, OFFSET равен 32**

```
        LOAD_I 0→0 (J)                          // 0x84000000 – адрес 0
        LOAD_I OFFSET+N-1→2                     // 0x8540002F – адрес 1
        LOAD_I N-2→3                            // 0x8460000E – адрес 2
        LOAD_I 0→10 (для сравнения с F)         // 0x86800000 – адрес 3
LJ:     LOAD_I 0→4 (F)                          // 0x84800000 – адрес 4
        LOAD_I OFFSET→1 (I)                     // 0x84200020 – адрес 5
LI:     LOAD_MI I→5                             // 0x94000025 – адрес 6
        INR I                                   // 0x0C000021 – адрес 7
        LOAD_MI I→6                             // 0x94000026 – адрес 8
        SUB 5,6→7 (5-6)                         // 0x080014C7 – адрес 9
        JC EI                                   // 0x50000010 – адрес 10
        DEC I                                   // 0x10000021 – адрес 11
        MOV_MI 6→I                              // 0x98000026 – адрес 12
        INR I                                   // 0x0C000021 – адрес 13
        MOV_MI 5→I                              // 0x98000025 – адрес 14
        LOAD_I 1→4 (F)                          // 0x84800001 – адрес 15
EI:     SUB 2,0→9 (OFFSET+N-1-J)                // 0x08000809 – адрес 16
        SUB I,9→9                               // 0x08000529 – адрес 17
        JNZ LI                                  // 0x44000006 – адрес 18
```

```
        SUB F,10→F (F-0)                    // 0x08001144 – адрес 19
        JZ END                              // 0x48000018 – адрес 20
        INR J                               // 0x0C000000 – адрес 21
        SUB J,3→9                           // 0x08000069 – адрес 22
        JNZ LJ                              // 0x44000004 – адрес 23
END:
```

Данный алгоритм сортировки был протестирован на следующем массиве:

19	54	23	25	74	3	11	34	87	41	21	7	33	14	66	10

В результате работы входной массив с данными был корректно отсортирован в порядке возрастания (рис. 6).

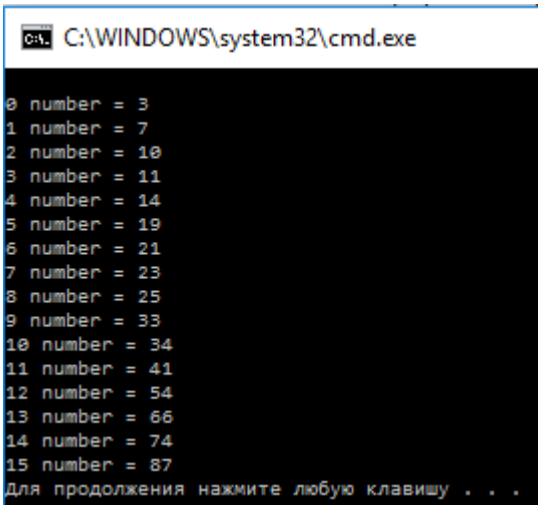

Рис. 6. Результат работы алгоритма сортировки массива.

2. Умножение матриц

```
        a[n][m], b[m][p], q[n][p] = 0
        LOAD_I offsetA→17                   // 0x86200030 – адрес 0
        LOAD_I offsetB→18                   // 0x86400044 – адрес 1
        LOAD_I offsetQ→19                   // 0x86600062 – адрес 2
        LOAD_I n→3                          // 0x84600004 – адрес 3
        LOAD_I m→4                          // 0x84800005 – адрес 4
        LOAD_I p→5                          // 0x84A00006 – адрес 5
        LOAD_I 0→0 (i)                      // 0x84000000 – адрес 6
LI:     LOAD_I 0→1 (j)                      // 0x84200000 – адрес 7
LJ:     MULT p,i→9 (indQ)                   // 0x140000A9 – адрес 8
        ADD indQ,j→ indQ (индекс q[i][j])   // 0x04002429 – адрес 9
        ADD indQ, offsetQ→indQ              // 0x04002669 – адрес 10
        LOAD_MI indQ →10 (q[i][j])          // 0x9400012A – адрес 11
        LOAD_I 0→2 (k)                      // 0x84400000 – адрес 12
```

```
LK:  MULT m,i→11 (indA)                  // 0x1400008B – адрес 13
     ADD indA,k→indA (индекс a[i][k])    // 0x04002C4B – адрес 14
     ADD indA, offsetA→indA              // 0x04002E2B – адрес 15
     LOAD_MI indA→12 (a[i][k])           // 0x9400016C – адрес 16
     MULT p,k→13 (indB)                  // 0x1400144D – адрес 17
     ADD indB,j→indB (индекс b[k][j])    // 0x0400342D – адрес 18
     ADD indB, offsetB→indB              // 0x0400364D – адрес 19
     LOAD_MI indB→14 (b[k][j])           // 0x940001AE – адрес 20
     MULT a,b→15 (a*b)                   // 0x140031CF – адрес 21
     ADD q,15→q (q+a*b)                  // 0x040029EA – адрес 22
     INR k                               // 0x0C000042 – адрес 23
     SUB k,m→16                          // 0x08000890 – адрес 24
     JNZ LK                              // 0x4400000D – адрес 25
     MOV_MI q→indQ                       // 0x9800012A – адрес 26
     INR j                               // 0x0C000021 – адрес 27
     SUB j,p→16                          // 0x080004B0 – адрес 28
     JNZ LJ                              // 0x44000008 – адрес 29
     INR I                               // 0x0C000000 – адрес 30
     SUB i,n→16                          // 0x08000070 – адрес 31
     JNZ LI                              // 0x44000007 – адрес 32
```

Алгоритм умножения матриц проверен на следующем примере: исходные данные – 2 матрицы, одна 4 на 5, другая 5 на 6, как результат умножения в итоге получается матрица 4 на 6, что говорит о корректности работы алгоритма (рис. 7).

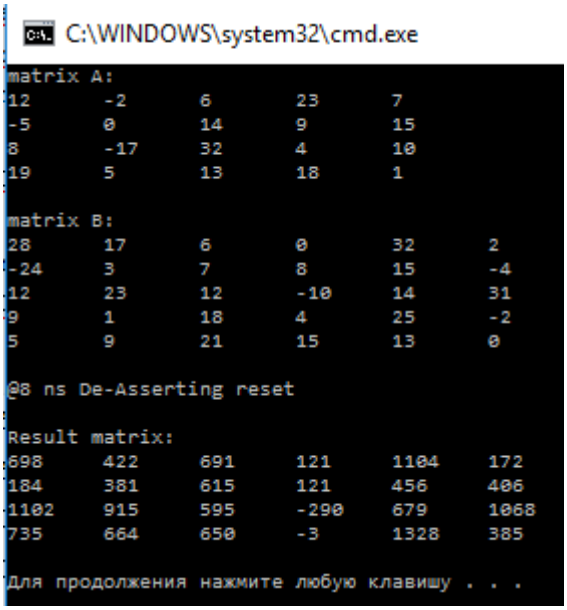

Рис. 7. Результат работы алгоритма умножения матриц.

3. Медианный фильтр

Алгоритм медианного фильтра:

```
allocate outputPixelValue[image width][image height]
   edgex := (window width / 2) rounded down
   edgey := (window height / 2) rounded down
   for x from edgex to image width - edgex
       for y from edgey to image height - edgey
           allocate colorArray[window width][window height]
           for fx from 0 to window width
               for fy from 0 to window height
                   colorArray[fx][fy] := inputPixelValue[x + fx - edgex][y + fy -
edgey]
           sort all entries in colorArray[][]
           outputPixelValue[x][y] := colorArray[window width / 2][window height /
2]
```

```
        LOAD_I image width→16                          // 0x86000005 – адрес 0
        LOAD_I image height→17                         // 0x86200007 – адрес 1
        LOAD_I window width→18                         // 0x86400003 – адрес 2
        LOAD_I window height→19                        // 0x86600003 – адрес 3
        LOAD_I offsetIn→20                             // 0x86800070 – адрес 4
        MULT i_width,i_height→21                       // 0x14004235 – адрес 5
        ADD offsetIn, 21→21 (offsetOut)                // 0x040052B5 – адрес 6
        MULT i_width,i_height→22                       // 0x14004236 – адрес 7
        ADD offsetOut,22→22 (offsetC)                  // 0x040056D6 – адрес 8
        SHIFT_R w_width→23 (edgex)                     // 0x1C000257 – адрес 9
        SHIFT_R w_height→24 (edgey)                    // 0x1C000278 – адрес 10
        SUB i_width, edgex→25 (для сравнения с x)      // 0x080042F9 – адрес 11
        SUB i_height, edgey→26 (для сравнения с y)     // 0x0800471A – адрес 12
        LOAD_I 0→x                                     // 0x87600000 – адрес 13
LX1:    LOAD_I 0→y                                     // 0x87800000 – адрес 14
LY1:    MULT i_height,x→12 (indOut)                    // 0x1400476C – адрес 15
        ADD indOut,y→indOut                            // 0x0400338C – адрес 16
        ADD indOut, offsetIn→15 (indIn)                // 0x0400328F – адрес 17
        ADD indOut,offsetOut→indOut                    // 0x040032AC – адрес 18
        LOAD_MI indOut→14 (in[][])                     // 0x940001EE – адрес 19
        MOV_MI in→indOut                               // 0x9800018E – адрес 20
        INR y                                          // 0x0C00039C – адрес 21
        SUB y,i_height→11                              // 0x0800722B – адрес 22
        JNZ LY1                                        // 0x4400000F – адрес 23
        INR x                                          // 0x0C00037B – адрес 24
```

```
        SUB x,i_width→11                          // 0x08006E0B – адрес 25
        JNZ LX1                                   // 0x4400000E – адрес 26
        MOV edgex→27 (x)                          // 0x880002FB – адрес 27
LX2:    MOV edgey→28 (y)                          // 0x8800031C – адрес 28
LY2:    LOAD_I 0→29 (fx)                          // 0x87A00000 – адрес 29
LFX:    LOAD_I 0→30 (fy)                          // 0x87C00000 – адрес 30
LFY:    MULT fx,w_height→15 (indC)                // 0x1400766F – адрес 31
        ADD indC,fy→indC (индекс cA[fx][fy])      // 0x04003FCF – адрес 32
        ADD indC,offsetC→indC                     // 0x04003ECF – адрес 33
        ADD x,fx→12 (indIn)                       // 0x04006FAC – адрес 34
        SUB indIn, edgex→indIn                    // 0x080032EC – адрес 35
        MULT indIn,i_height→indIn                 // 0x1400322C – адрес 36
        ADD indIn,y→indIn                         // 0x0400338C – адрес 37
        ADD indIn,fy→indIn                        // 0x040033CC – адрес 38
        SUB indIn,edgey→indIn (индекс in[][])     // 0x0800330C – адрес 39
        ADD indIn,offsetIn→indIn                  // 0x0400328C – адрес 40
        LOAD_MI indIn→13 (in[][])                 // 0x9400018D – адрес 41
        MOV_MI in→indC                            // 0x980001ED – адрес 42
        INR fy                                    // 0x0C0003DE – адрес 43
        SUB fy, w_height→11                       // 0x08007A6B – адрес 44
        JNZ LFY                                   // 0x4400001F – адрес 45
        INR fx                                    // 0x0C0003BD – адрес 46
        SUB fx, w_width→11                        // 0x0800764B – адрес 47
        JNZ LFX                                   // 0x4400001E – адрес 48
        LOAD_I 0→8 (s)                            // 0x85000000 – адрес 49
        MOV w_height→3 (для подпрограммы)         // 0x88000263 – адрес 50
        DEC 3→3                                   // 0x10000063 – адрес 51
        DEC 3→3 (N-2)                             // 0x10000063 – адрес 52
        MOV offsetC→15                            // 0x880002CF – адрес 53
        MOV offsetC→2                             // 0x880002C2 – адрес 54
        ADD 2,3→2                                 // 0x04000862 – адрес 55
        INR 2→2 (offset+N-1)                      // 0x0C000042 – адрес 56
LS:     CALL sort                                 // 0x54000050 – адрес 57
        ADD 15, w_height→15 (offsetSort)          // 0x04003E6F – адрес 58
        ADD 2, w_height→2 (offsetSort+N-1)        // 0x04000A62 – адрес 59
        INR s                                     // 0x0C000108 – адрес 60
        SUB s,w_width→11                          // 0x0800224B  – адрес 61
        JNZ LS                                    // 0x44000039 – адрес 62
        MULT i_height,x→12 (indOut)               // 0x1400476C – адрес 63
```

```
        ADD indOut,y→indOut                  // 0x0400338C – адрес 64
        ADD indOut,offsetOut→indOut          // 0x040032AC – адрес 65
        MULT w_height, edgex→indC            // 0x14004EEF – адрес 66
        ADD indC, edgey→indC                 // 0x04003F0F – адрес 67
        ADD indC,offsetC→indC                // 0x04003ECF – адрес 68
        LOAD_MI indC→14 (c[][])              // 0x940001EE – адрес 69
        MOV_MI c→indOut                      // 0x9800018E – адрес 70
        INR y                                // 0x0C00039C – адрес 71
        SUB y, 26→11                         // 0x0800734B – адрес 72
        JNZ LY2                              // 0x4400001D – адрес 73
        INR x                                // 0x0C00037B – адрес 74
        SUB x,25→11                          // 0x08006F2B – адрес 75
        JNZ LX2                              // 0x4400001C – адрес 76
```

В качестве подпрограммы использовалась ранее разработанная программа сортировки, но она была немного изменена:

```
        LOAD_I 0→0 (J)                       // 0x84000000 – адрес 80
        LOAD_I 0→10 (для сравнения с F)      // 0x85400000 – адрес 81
LJ:     LOAD_I 0→4 (F)                       // 0x84800000 – адрес 82
        MOV 15→1 (I)                         // 0x880001E1 – адрес 83
LI:     LOAD_MI I→5                          // 0x94000025 – адрес 84
        INR I                                // 0x0C000021 – адрес 85
        LOAD_MI I→6                          // 0x94000026 – адрес 86
        SUB 5,6→7 (5-6)                      // 0x080014C7 – адрес 87
        JC EI                                // 0x5000005E – адрес 88
        DEC I                                // 0x10000021 – адрес 89
        MOV_MI 6→I                           // 0x98000026 – адрес 90
        INR I                                // 0x0C000021 – адрес 91
        MOV_MI 5→I                           // 0x98000025 – адрес 92
        LOAD_I 1→4 (F)                       // 0x84800001 – адрес 93
EI:     SUB 2,0→9 (OFFSET+N-1-J)             // 0x08000809 – адрес 94
        SUB I,9→9                            // 0x08000529 – адрес 95
        JNZ LI                               // 0x44000054 – адрес 96
        SUB F,10→F (F-0)                     // 0x08001144 – адрес 97
        JZ END                               // 0x48000066 – адрес 98
        INR J                                // 0x0C000000 – адрес 99
        SUB J,3→9                            // 0x08000069 – адрес 100
        JNZ LJ                               // 0x44000052 – адрес 101
END: RET                                     // 0x58000000 – адрес 102
```

Работа алгоритма проверялась на исходной матрице 5 на 7, в результате получается отфильтрованная матрица такой же размерности (рис. 8).

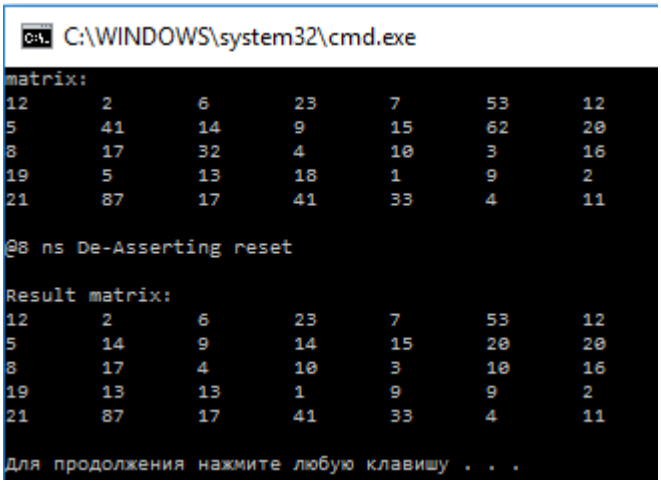

Рис. 8. Результат работы алгоритма медианного фильтра.