Министерство образования и науки РФ

Санкт-Петербургский Политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа искусственного интеллекта

Направление «Математика и компьютерные науки»

Отчёт о прохождении производственной практики «Разработка специализированного процессора для сортировки массива алгоритмом гномьей сортировки»

Студент: Кулыгин Е. А. гр. 3530201/00001

Руководитель: Чуватов Михаил Владимирович

Санкт – Петербург 2022

Содержание

Введение

В данном отчете представлено описание этапов проектирования специализированного процессора для решения задачи сортировки массива, состоящих из целочисленных положительных элементов разрядностью от 8 до 32 бит.

Архитектура процессора и набор команд разработаны для выполнения узкого набора операций. При разработке специализированного процессора были учтены требования и ограничения заказчика.

Работа должна быть выполнена в графическом инструменте для разработки и моделирования логических схем Logisim.

1. Постановка задачи

Цель работы: разработать архитектуру набора команд, которые позволят отсортировать одномерный массив целочисленных значений методом гномьей сортировки.

Разработанную архитектуру реализовать в модели компьютера.

Задачи:

- Определить особенности алгоритма гномьей сортировки.
- Сформулировать необходимый для реализации набор команд.
- На основе примерного набора команд понять какие элементы следует использовать в схеме и как осуществлять связь между элементами; как реализовать переход от одного элемента массива к другому; как менять элементы массива местами; как переходить от одной команды к другой по условию.
- Для составленной схемы продумать какие операции следует выполнять в командах для их корректной работы; понять каким образом следует хранить данные и команды в памяти; составить итоговый набор команд с кодировками для построенной схемы, а также разработать программу с использованием этих команд, которая будет выполнять сортировку массива методом пузырька и выводить значения после сортировки на индикаторы.
- Полученную программу протестировать на нескольких наборах данных.

Ограничения:

• Ограничение исходных данных:

– Одномерный массив произвольных целых чисел.
– Диапазон принимаемых значений каждого элемента от -32768 до 32767.
– Количество элементов от 2 до 64.

• Ограничения среды моделирования:

– Разрядности параллельных интерфейсов (шин) – не более 16 бит.
– Разрядность командного регистра – не более 16 бит.
– Разрядность адресного регистра – не более 12 бит.
– Разрядность ячейки ОЗУ – не более 16 бит.
– Разрядности регистров общего назначения, сумматора, логических элементов – не более 8 бит.
– Принстонская архитектура (общее ОЗУ для команд и данных, общая шина для команд, адресов, данных).
– Допустимые компоненты: разветвитель, контакт, тоннель, согласующий резистор, тактовый генератор, константа (только в устройстве вывода на индикаторы), расширитель

битов, все элементы, кроме «нечётность» и «чётность», декодер, селектор битов, сумматор, делитель (только в устройстве вывода на индикаторы), все элементы памяти, кроме «сдвиговый регистр» и «генератор случайных чисел», 7-сегментный индикатор, шестнадцатеричный индикатор.

2. Введение в предметную область

2.1 Принстонская архитектура

Принстонская архитектура или архитектура фон Неймана — широко известный принцип совместного хранения команд и данных в памяти компьютера.

Принципы архитектуры фон Неймана:

- Принцип двоичности.

  Для представления данных и команд используется двоичная система счисления.

- Принцип программного управления.

  Программа состоит из набора команд, которые выполняются процессором друг за другом в определённой последовательности.

- Принцип однородности памяти.

  Как команды, так и данные хранятся в одной и той же памяти. Над командами можно выполнять такие же действия, как и над данными.

- Принцип адресуемости памяти.

  Структурно основная память состоит из пронумерованных ячеек; процессору в произвольный момент времени доступна любая ячейка.

- Принцип последовательного программного управления.

  Все команды располагаются в памяти и выполняются последовательно, одна после завершения другой.

- Принцип условного перехода.

  Команды из программы не всегда выполняются одна за другой. Возможно присутствие в программе команд условного перехода, которые изменяют последовательность выполнения команд в зависимости от значений данных.

2.2 Гномья сортировка

Разработанный набор команд будет предназначен для решения задачи сортировки массива, состоящего из целочисленных элементов в диапазоне от -32768 до 32767. Гномья сортировка — алгоритм сортировки, похожий на сортировку вставками, но в отличие от последней перед вставкой на нужное место происходит серия обменов, как в сортировке пузырьком. Название происходит от предполагаемого поведения садовых гномов при сортировке линии садовых горшков.

Гномья сортировка основана на технике, используемой обычным голландским садовым гномом (нидерл. tuinkabouter). Это метод, которым садовый гном сортирует линию цветочных горшков. По существу он смотрит на текущий и предыдущий садовые горшки: если они в правильном порядке, он шагает на один горшок вперёд, иначе он меняет их местами и шагает на один горшок назад. Граничные условия: если нет предыдущего горшка, он шагает вперёд; если нет следующего горшка, он закончил.

Вход: массив array[0..N]
Выход: отсортированный по неубыванию массив array[0..N]
Рис. 1: Описание алгоритма сортировки посредством псевдокода

```
gnomeSort(a[0..size - 1])
   i = 1;
   j = 2;
   while i < size
      if a[i - 1] > a[i] //для сортировки по возрастанию поменяйте знак сравнения на <
         i = j;
         j = j + 1;
      else
         swap a[i - 1] and a[i]
         i = i - 1;
         if i == 0
            i = j;
            j = j + 1;
```

3. Особенности реализации

3.1 Описание набора команд

Для реализации алгоритма гномьей сортировки был разработан набор из 13 команд. Продолжительность выполнения отдельной команды 8 тактов.

Продолжительность выполнения команд 5 тактов.

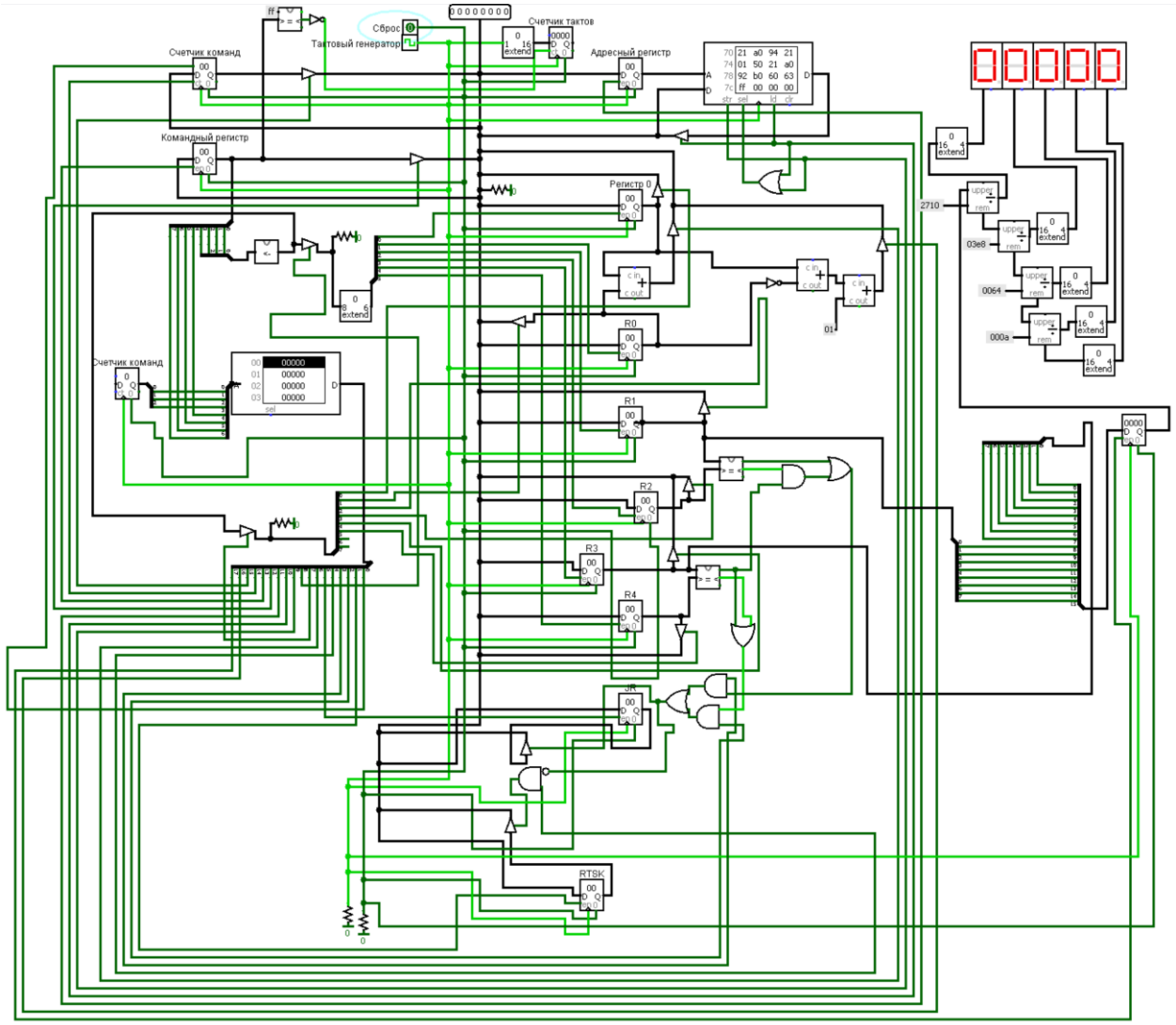
1. NOP – загружает команду по адресу из счетчика команд в регистр команд.
2. LDR Rn [mem] – сохраняет в регистр Rn значение ячейки памяти, адрес которой хранится в операнде.
3. LDI Rn val – сохраняет в регистр Rn значение переменной.
4. STR Rn [mem] – сохраняет значение регистра Rn по адресу в памяти.
5. SUB Rn – сохраняет в регистр Rn разность значений, лежащих в регистрах R0 и R1.
6. ADD Rn – сохраняет в регистр Rn сумму значений, лежащих в регистрах R0 и R1.
7. UCJ Rn [mem] – команда безусловного перехода. Реализует безусловный переход к команде по адресу в памяти.
8. CJ1 [mem] – условный переход к команде по адресу в памяти, если выполнено условие, что первый элемент больше следующего за ним элемента
9. CJ2 [mem] – условный переход к команде по адресу в памяти, если выполнено условие, что элемент 1 больше элемента 2, либо равен ему. Данная команда реализована для проверки выполнения цикла while.
10. VAL Rn – сохраняет в регистр Rn значение по адресу. Адрес представляет собой сумму операндов, лежащих в регистрах R0 и R1
11. VALS Rn – сохраняет в памяти значение регистра Rn по адресу, который является суммой значений регистров R0 и R1.
12. OUT – выводит на экран значение регистра вывода.
13. STOP – при выполнении данной команды выключается счетчик команд, в командный регистр перестают записываться новые значения.

3.2 Общая схема и её особенности

Схема состоит из 8-битной шины, через которую происходит обмен данными между компонентами схемы, тактового генератора и точки сброса, которая нажимается вручную.
Всего схему можно разделить на шесть компонент: счётчик команд и ОЗУ, система декодирования с ПЗУ, регистры общего назначения А и Б и связанные с ними сумматор и
логическое устройство сравнения, регистры общего назначения 0-4 и связанные с

ними логические устройства сравнения, регистр флагов и регистр перехода и связанное с ними логическое устройство перехода, а так же подсистема вывода, состоящая из индикаторов, двух регистров и устройства вывода значения.

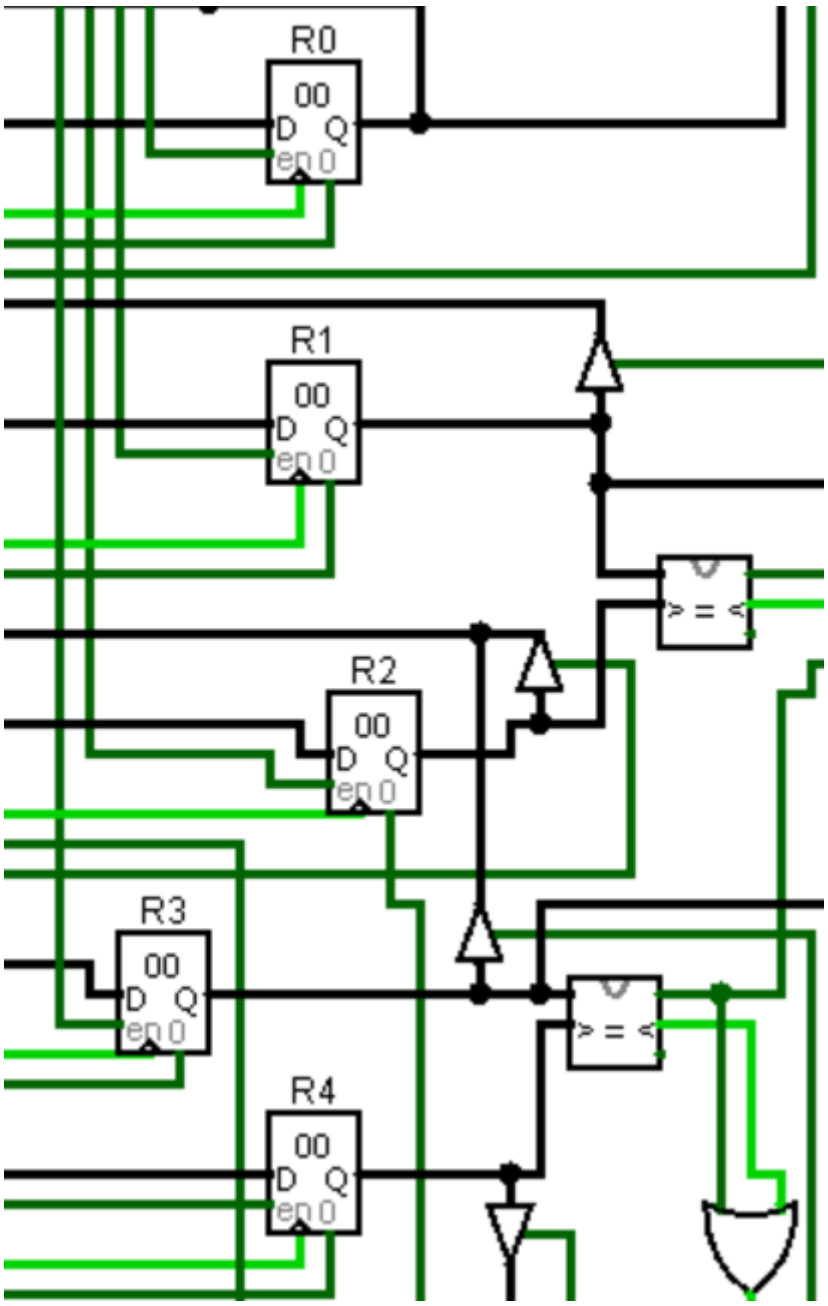
При работе компьютера команды извлекаются по адресу, хранящемуся в счетчике команд. Сохранение значений в регистры общего назначения, адресный регистр, счетчик команд и командный регистр реализуется с помощью управляющих сигналов, которые осуществляют вывод данных на шину и запись в необходимый регистр. Также из-за ограничений на разрядность регистров при сравнении двух 16-битных значений первое число загружается в регистры R2 и R4, в регистр R2 - старший байт, в регистр R4 - младший байт, а второе в регистры R3 и R5 - в регистр R3 - старший байт, в регистр R5 - младший байт.



3.3 Регистры общего назначения

На разработанной схеме присутствуют пять регистров общего назначения: R0, R1, R2, R3, R4, R5. Разрядность регистров 8 бит. Входы и выходы регистров общего назначения управляются декодером, значения передаются по общей шине.

Входы регистров JR и RTSK управляются с помощью управляющих сигналов. Выходы этих регистров управляются подсистемой условных переходов.


R0
00
D Q
en 0
R1
00
D Q
en 0
> = <
R2
00
D Q
en 0
R3
00
D Q
en 0
> = <
R4
00
D Q
en 0

3.4 Подсистема условных переходов

Подсистема условных переходов состоит из двух частей:

1) первая часть реализует операцию сравнения операндов с помощью построенного компаратора, лежащих в регистрах R2-R5

2) вторая часть загружает необходимый адрес команды в счетчик команд.

Данная система поддерживает два вида условных переходов и один безусловный.

При реализации подсистемы условных переходов адрес команды, к которой необходимо перейти, сохраняется в регистр JR. Управляющий сигнал передает значение регистра JR на шину, затем адрес сохраняется в счетчик команд.

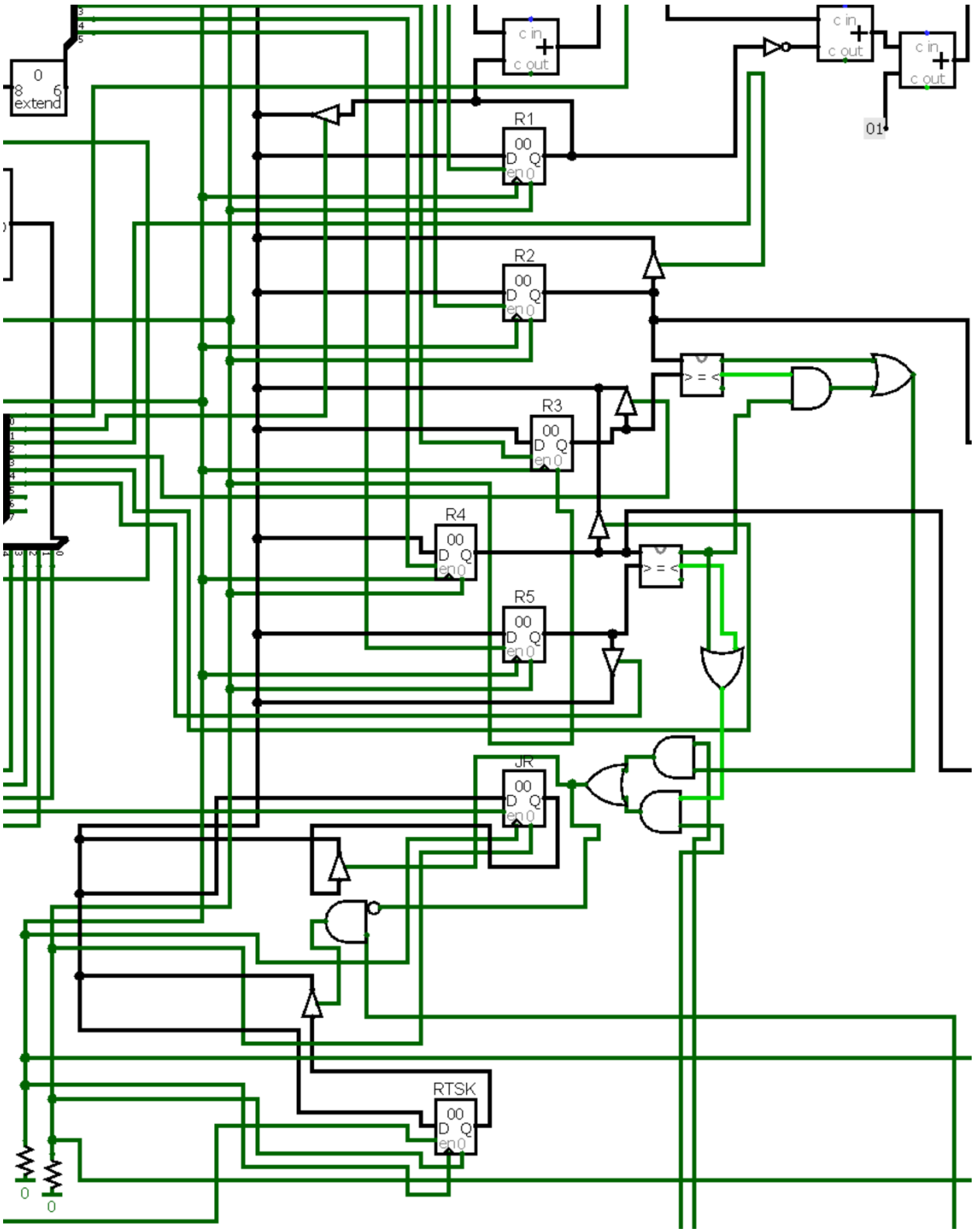
Есть два условия: условие <<больше>> и условие <<больше или равно>>.

Нас интересуют условие цикла <<если i больше или равно n>> и условие <<строго больше>>, относящееся к элементам массива.

Если значение (i-1)-го элемента массива больше значения i-го элемента, то срабатывает флаг, и на шину с помощью управляемого буфера поступает адрес, по которому нужно перейти, и записывается в счетчик команд.

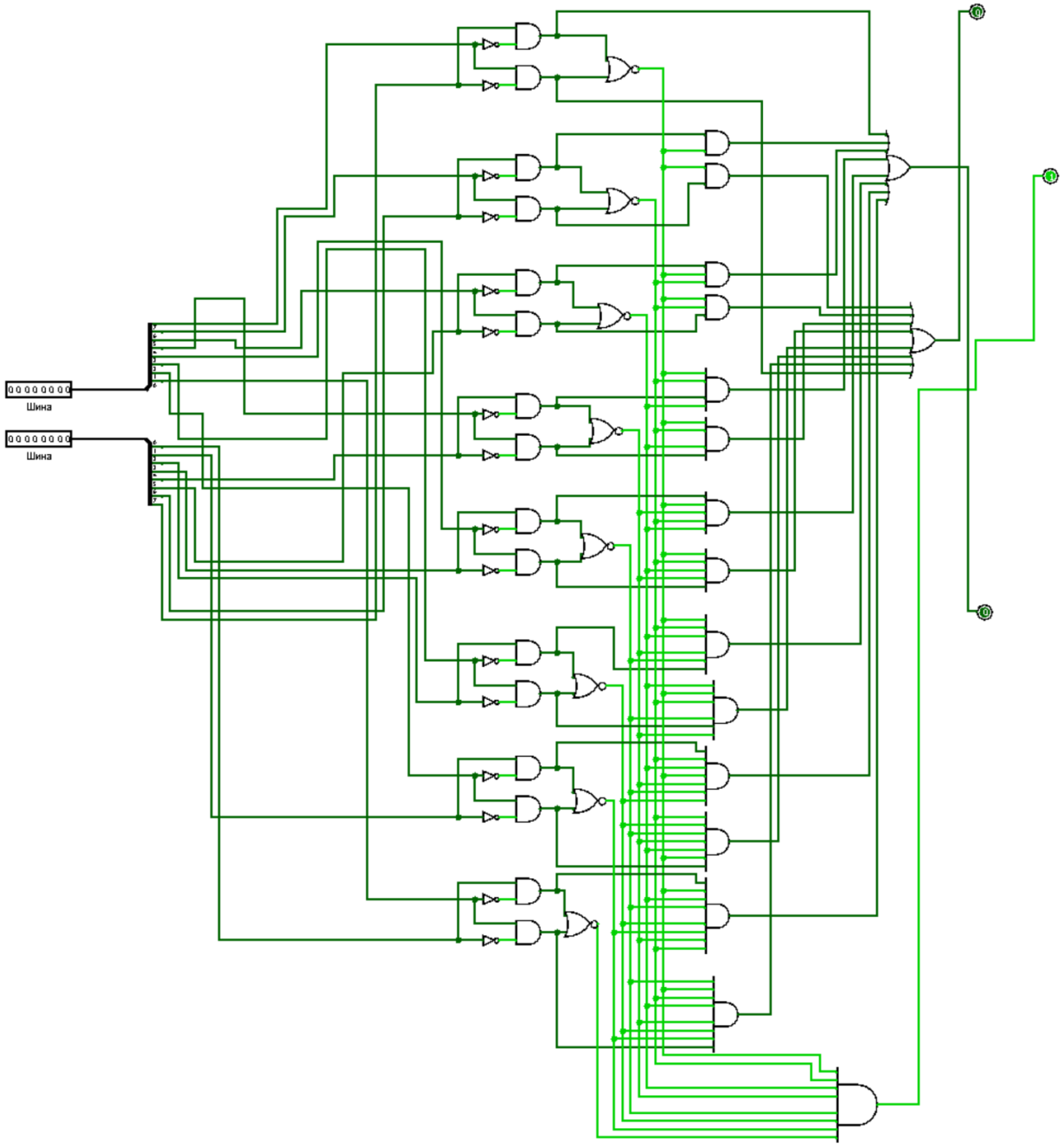
В зависимости от срабатывания флага на шину выпускается либо значение, либо адрес по которому нужно перейти в случае не выполнения условия, либо адрес по которому нужно перейти, если условие выполняется. При не выполнении условия, срабатывает инвертор и на шину выпускается текущее значение счетчика команд. Инвертированное значение и управляющий сигнал дают в логическом И 1 и включают управляемый буфер для регистра, который хранит текущее значение счётчика. Если не реализовывать данную логику, то может потеряться значение счетчика команд.

Условие <<больше или равно>> - второй вариант условного перехода. Проверка на прекращение цикла. Если использовать для сравнения элементов условие <<больше или равно>>, то цикл может превратиться в бесконечный в случае, если в элемента равны и постоянно меняются местами.


extend
c in
c out
R1
R2
R3
R4
R5
JR
RTSK
00
D Q
en 0
> = <
01
0

3.5 Компаратор

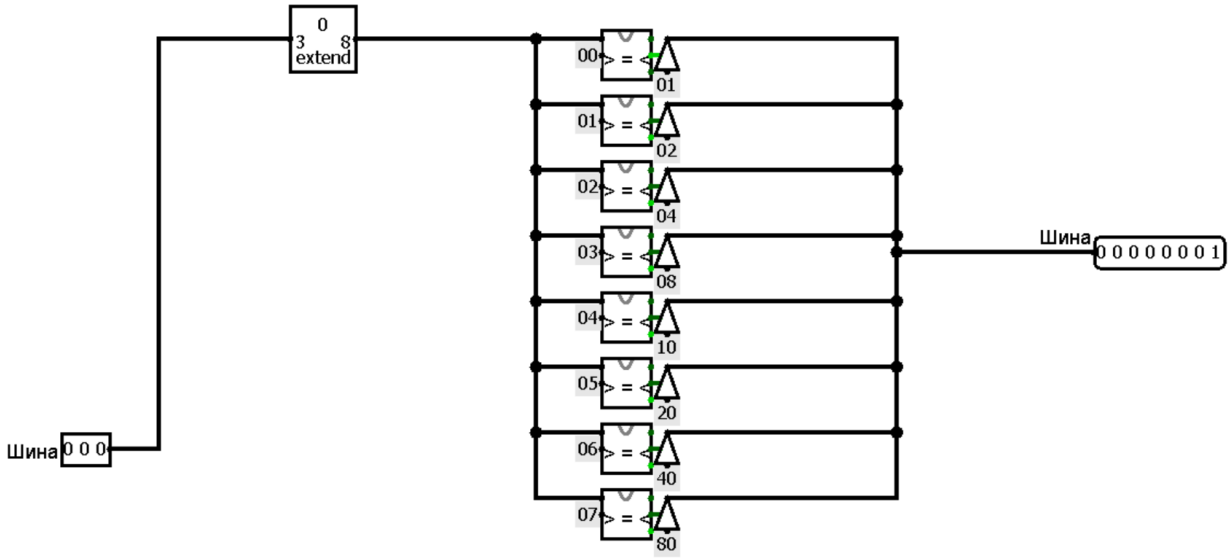
Компаратор реализован в дополнительной схеме. Он принимает на вход два 8 битных значения и выводит один из трех сигналов. Больше, равно и меньше соответственно. Компараторы используются в сравнении значений регистров, в схеме шифтера и при работе счетчика тактов. Схема компаратора приведена на рисунке.



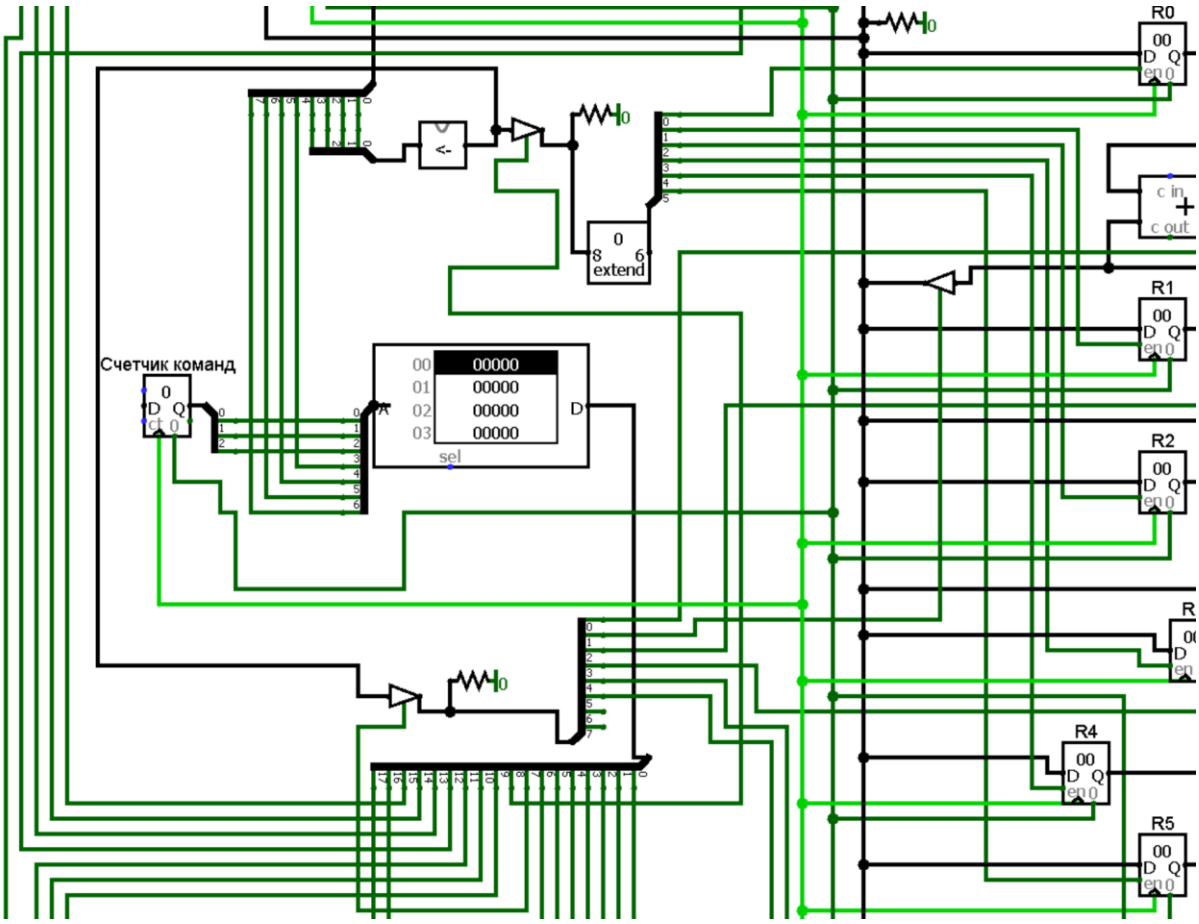
Внешний вид компаратора на схеме.

3.6 Шифтер (сдвигатель)

Сдвигатель используется в схеме единожды, для оптимизации записи значений в регистры общего назначения. Сдвигатель получает на вход 3-битное значение обозначающее порядковый номер регистра общего назначения и затем сдвигает единицу влево на то значение, которое поступает на вход. Таким образом при реализации декодера мы не должны добавлять 5 дополнительных выходов на разветвитель. Реализация сдвигателя приведена на рисунке.



Использование сдвигателя в схеме main, также приведено на рисунке.



Внешний вид сдвигателя на схеме.

3.7 ОЗУ и счётчик команд

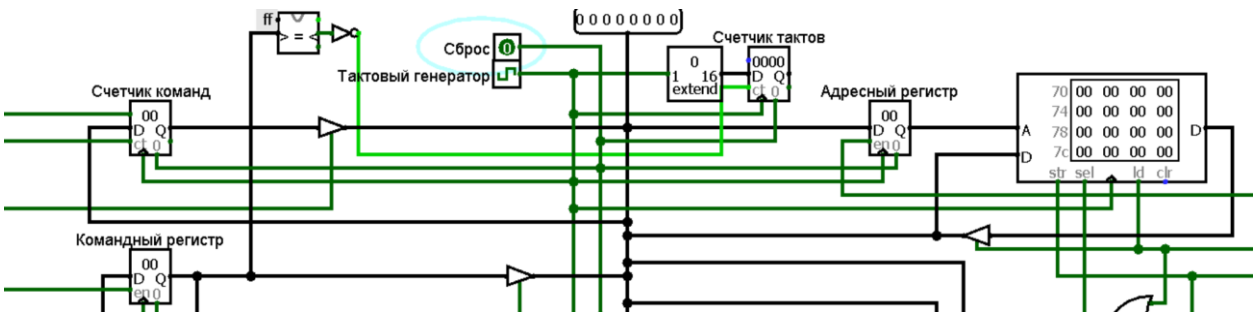
Для работы с ОЗУ используются управляющие сигналы: вход адресного регистра, чтение из памяти, запись в память и очистка памяти (используется вручную).
ОЗУ хранит закодированную последовательность команд, а также все необходимые данные
для реализации сортировки массива методом пузырька.
Команды и операнды хранятся в одной области памяти. Разрядность данных составляет 8 бит, разрядность адреса — 8 бит.

Доступ к командам в ОЗУ происходит через счётчик команд — 8-битный регистр, который содержит адрес в ОЗУ конкретной команды. Для перехода к следующей команде счётчик команд инкрементируется, для безусловного и условного переходов в счётчик команд заносится
адрес команды в ОЗУ.
Для чтения команды из оперативной памяти в адресный регистр записывается значение счётчика команд. Затем по записанному адресу считывается команда и записывается в командный регистр.
Обмен данными происходит посредством 8-битной шины.

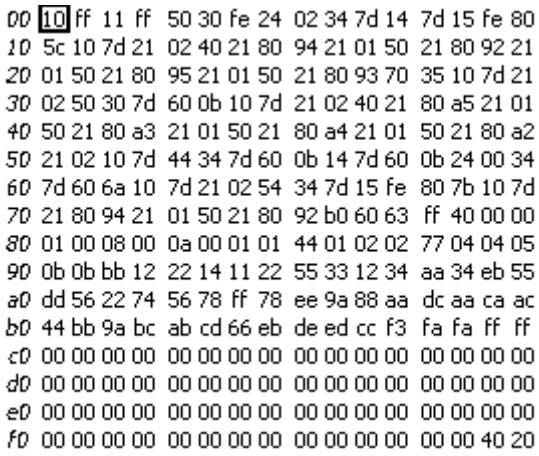
- Программа сортировки хранится в ячейках памяти 0x00-0x7c.
- Значение счетчика хранится по адресу 0x7d.
- Элементы массива хранятся в ячейках, начиная с 0x80 и заканчивая 0xfd, в зависимости от количества элементов в массиве.
- Количество элементов в сортируемом массиве лежит по адресу 0xff.
- В оставшиеся ячейки памяти могут быть занесены дополнительные элементы массива, общее результирующее количество элементов массива необходимо указать в ячейку 0xff.

Для корректной работы программы пользователю необходимо вписать количество элементов массива в ячейку 0xff. Также пользователю нужно вписать элементы массива в ячейки памяти, начиная с 0x80 и заканчивая 0xfd, в зависимости от количества элементов в массиве. Доступ к ячейкам памяти, в которых хранятся команды, осуществляется с помощью счетчика команд, значение которого записывается в адресный регистр.

Схема связи счетчика команд и ОЗУ указана на рисунке.



Размещение элементов массива в ОЗУ: элементы массива должны располагаться последовательно, друг за другом. Сначала в ячейку с младшим адресом располагаются старшие 8 бит числа, занимая младшие 8 бит ячейки. Затем в следующую ячейку заносятся оставшиеся младшие 8 бит, занимая те же младшие 8 бит ячейки. Таким образом элемент массива занимает две подряд идущие ячейки памяти ОЗУ. Организация ОЗУ с 32 битными элементами указана на рисунке.

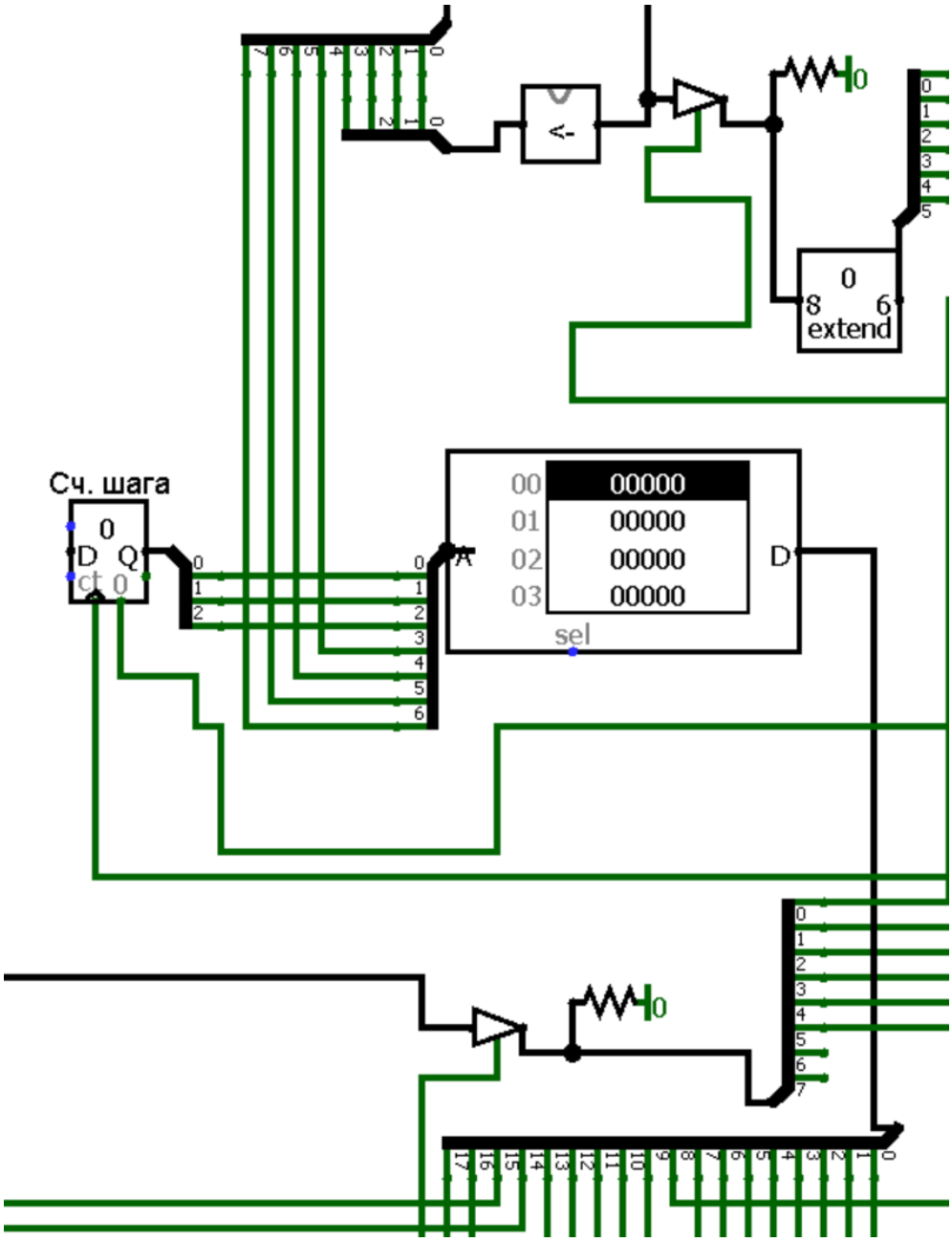


3.8 Декодер

Реализован с помощью командного регистра, постоянного запоминающего устройства, счетчика шагов, управляющих сигналов и системы обработки номера регистра.

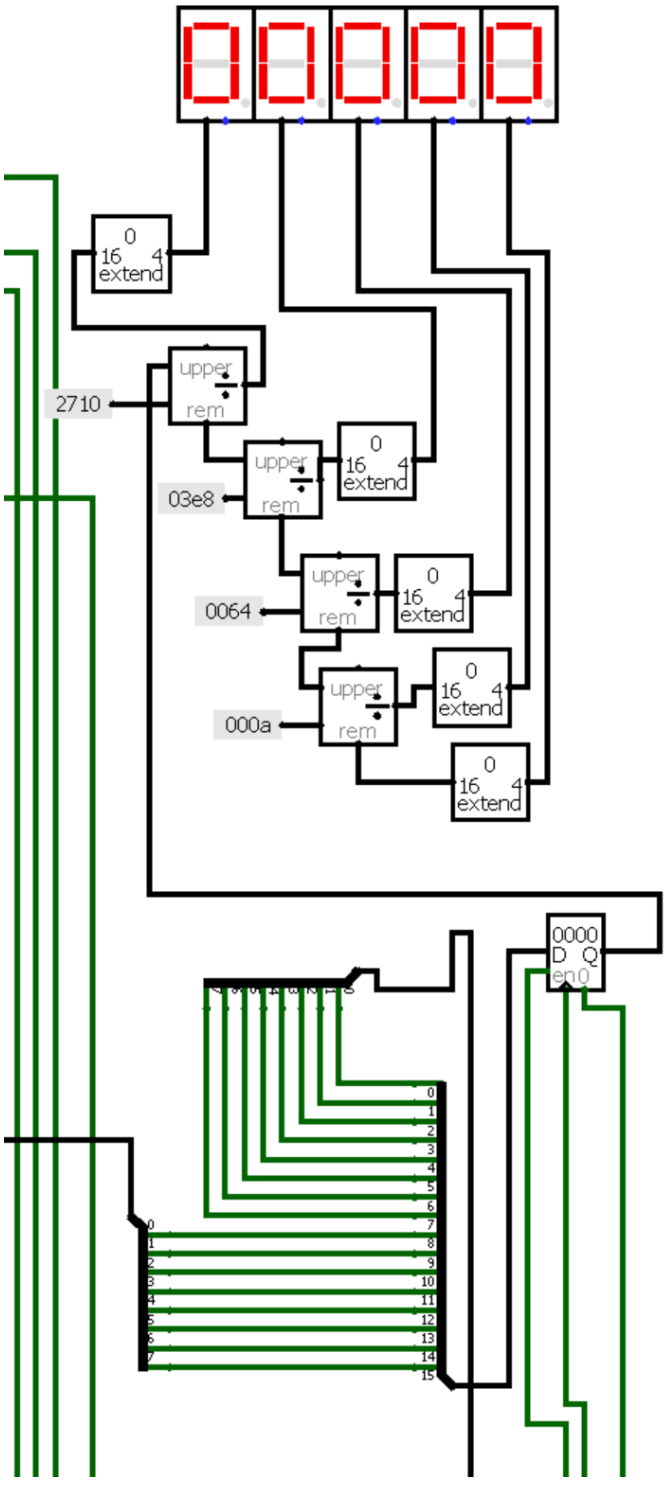
Разрядность адреса: 7 бит. Старшие 4 бита представляют собой код команды и подаются на вход декодера из командного регистра. Младшие три бита формируются из значения в счетчике шагов. Разрядность данных декодера: 18 бит. С помощью декодера осуществляется подача 18 управляющих сигналов: вывод результата, вычитание, выход счетчика команд, включение счетчика команд, вход командного регистра, выход командного регистра, вход адресного регистра, чтение из памяти, запись в память, вход регистра Rn, выход регистра Rn, выход сумматора, вход регистра JR, выход регистра JR, флаг1, флаг2, вход регистра RTSK, включение счетчика команд.

Работа с регистром, указанным в коде команды, осуществляется с помощью сдвигателя битов. Младшие четыре бита команды - значение, на которое сдвигается единица влево. Выход сдвигателя - управляющий сигнал на вход и выход значений регистра.



3.9 Система вывода

На рисунке представлена подсистема вывода результата. Подсистема вывода состоит из 5 шестнадцатиричных индикаторов и шестнадцатибитного регистра. Данный регистр с помощью разветвителя соединен с восьмиразрядными регистрами R2 и R4, которые содержат старшую и младшую часть элемента соответственно. Также подистема вывода содержит 4 делителя, с помощью которых значение элемента массива можно представить пользователю в виде десятичного числа.



3.10 Описание программы

Программа, реализующая гномью сортировку массива и вывод элементов массива на дисплей содержит 72 команды.

Для корректной работы программы пользователю необходимо вписать количество элементов массива в ячейку 0xff. Также пользователю нужно вписать элементы массива в ячейки памяти, начиная с 0x80 и заканчивая 0xf9, в зависимости от количества элементов в массиве

Загрузка значений, доступ к переменным и их изменение осуществляется с помощью команд LDR, LDI и STR. С помощью команды VAL осуществляется сохранение в регистры R2-R5 результатов операций над элементами массива. В Регистре R0 хранится номер необходимого элемента, в регистре R1 хранится адрес нулевого элемента массива. Подробное описание кода программы, реализующей алгоритм гномьей сортировки см. Приложение 1.

4. Тестирование

4.1 Требования

• Необходимо протестировать разработанную схему, набор команд и программу на правильность сортировки пузырьком массива элементов в диапазоне от -32768 до 32767. Количество элементов массива должно находиться в диапазоне от 2 до 64 элементов.
• Выявить зависимость времени выполнения от исходного числа элементов массива и сравнить полученную зависимость с теоретической сложностью выполнения алгоритма пузырьковой сортировки.
• При подсчёте времени и тактов не учитывать часть программы, которая выводит элементы массива на индикаторы.
16
• При проведении тестирования убедиться в том, что никакие элементы схемы не вызывают ошибок, например ошибки ввода данных на шину, когда несколько компонент схемы пытаются положить одновременно данные на шину, и работают слаженно между собой.

4.2 Проведение тестирования

Было подготовлено 3 массива разной размерности: 8, 16, 32.
Первый набор данных был собран вручную, следуя ограничениям для проверки правильности работы реализованного логического устройства сравнения двух 16-тиричных значений..

диапазона от -32768 до 32767, при этом в массиве не могло быть исключительно положительных или отрицательных чисел.

В результате тестирования были получены следующие данные:
• 1 Набор данных
Массив 8 значений
Сложность алгоритма: $8^2 = 64$
Количество тактов: 6870
Время: 6870/4100 = 1,675с
• 2 Набор данных
Массив 16 значений
Сложность алгоритма: $16^2 = 256$
Количество тактов: 26225
Время: 26225/4100 = 6,396 с

• 3 Набор данных
Массив 32 значения
Сложность алгоритма: $32^2 = 1024$
Количество тактов: 68248
Время: 68248/4100 = 16,645 с

4.3 Результаты тестирования

Тестирование прошло успешно для всех 4 наборов данных: элементы всех массивов, после выполнения программы, были расположены в порядке возрастания, как и требовалось.
Ошибок выявлено не было.
Анализ зависимости времени выполнения от числа элементов исходного массива

Количество элементов	Время выполнения сортировки,с
8	1,675
16	6,396
32	16,645

Таблица 1.

Из таблицы 1 видно, что для количества элементов массива, которые отличаются между собой в 2 раза время выполнения вырастает примерно в 4 раза. То есть мы можем отчетливо отследить зависимость близкую к квадратичной, что как раз-таки и должно получаться согласно теоретическому значению $O(n^2)$.

Заключение

В результате данной научно-исследовательской работы был разработан специализированный процессор для сортировки массивов от 2 до 64 целочисленных 16-битных значений алгоритмом гномьей сортировки. Была реализована система команд. А также процессор для обработки данных программ. Результатом работы стал проект в Logisim представляющий собой процессор для сортировки массивов с помощью гномьей сортировки.
Преимуществом данного процессора является ориентирование на конкретно поставленную задачу, что позволяет ему оптимально использовать память исключительно на необходимые действия. При сортировке 64 размерного массива используются абсолютно все ячейки ОЗУ. Это означает, что память была реализована наиболее подходящим образом.
Команды и данные хранятся в единой последовательно адресуемой памяти, что соответствует архитектуре фон Неймана, используется одна шина, на которую передаются и данные, и команды.

Процессор может быть масштабирован до работы с массивами большего размера, но для этого необходимо расширить разрядность ОЗУ. И изменить разрядности многих блоков.

Также, можно оптимизировать схему исключив из нее счетчик тактов и демонстрирующий экран вывода.

Программа была протестирована на нескольких массивах различной длины и во всех случаях успешно завершала сортировку. Никаких ошибок не было выявлено. Также, определена зависимость работы процессора близкая к теоретической квадратичной зависимости $O(n^2)$.

Список литературы:

[1] Таненбаум, Э. Архитектура компьютера / Э. Таненбаум, Т. Остин ; перевод с английского Е. Матвеева. — 6-е изд. — Санкт-Петербург [и др.] : Питер, 2014.

[2] Logisim: Документация. http://cburch.com/logisim/ru/docs.html(дата обращения: 01.07.2022).

Приложение 1

[sec:A]

```
10 ff | LDR r0 ff
11 ff | LDR r1 ff
50    | ADD r0
30 fe | STR r0 fe
24 02 | LDI r4 02      i=1
34 7d | STR r4 7d
14 7d | LDR r4 7d      while (i<n)
15 fe | LDR r5 fe
80 5c | CJ2 5c
10 7d | LDR r0 7d      if(ar[i-1]<=ar[i])
21 02 | LDI r1 02
40    | SUB r0
21 80 | LDI r1 a0
94    | VAL r4
21 01 | LDI r1 01
50    | ADD r0
21 a0 | LDI r1 a0
92    | VAL r2
21 01 | LDI r1 01
50    | ADD r0
21 a0 | LDI r1 a0
95    | VAL r5
21 01 | LDI r1 01
50    | ADD r0
21 a0 | LDI r1 a0
93    | VAL r3
70    | CJ1 35
10 7d | LDR r0 7d      i++
21 02 | LDI r1 02
50    | ADD r0
30 7d |STR r0 7d

60 ob | UCJ ob
10 7d | LDR r0 7d      else
21 02 | LDI r1 02
40    | SUB r0
21 80 | LDI r1 a0
a5    | VALS r5
21 01 | LDI r1 01
50    | ADD r0
21 80 | LDI r1 a0
a3    | VALS r3
21 01 | LDI r1 01
50    | ADD r0
21 80 | LDI r1 a0
a4    | VALS r4
```

```
21 01 | LDI r1 01
50    | ADD r0
21 80 | LDI r1 a0
a2    | VALS r2
21 02 | LDI r1 02   i- -
10 7d | LDR r0 7d
44    | SUB r4
34 7d | STR r4 7d

60 0b | UCJ 0b
24 00 | LDI r4 00    for(i=0; i<N; i++)
34 7d | STR r4 7d
60 6a | UCJ 6a
10 7d | LDR r0 7d
01 02 | LDI r1 02
54    | ADD r4
34 7d | STR r4 7d
15 fe | LDR r5 fe
80 7b | CJ2 7b
10 7d | LDR r0 7d     cout<<arr[i]
21 80 | LDI r1 80
94    | VAL r4
21 02 | LDI r1 01
50    | ADD r0
21 80 | LDI r1 80
92    | VAL r2
b0    | OUT
60 63 | UCJ 63
```