МИНОБОРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт прикладной математики и механики

Кафедра «Телематика (при ЦНИИ РТК)»

Отчет по производственной практике

Разработка специализированного процессора для сортировки массива алгоритмом плавной сортировки

Обучающийся: Сеннов В.Н. гр. 3630201/80002 _____

Руководитель: Чуватов М.В. _____

Санкт-Петербург
2020

Содержание

Введение

Особенности разработки специального аппаратного обеспечения под требования заказчика

В отличие от процессоров общего назначения, специальные процессоры имееют более узкие функции, но выполнение этих функций должно происходить эффективнее. Архитектура процессора, набор команд должны быть разработаны для выполнения узкого набора алгоритмов. Также при разработке необходимо учитывать требования и ограничения со стороны заказчика, например, ограничения на элементную базу.

Алгоритм плавной сортировки

Плавная сортировка является разновидностью пирамидальной сортировки. Алгоритм был разработан Э. Дейкстрой.

Идея алгоритма заключается в формировании из элементов массива специальной структуры данных — кучи. Формирование кучи происходит быстро, после того как куча сформирована, из нее легко выделяется максимальный элемент. После удаления максимального элемента куча формируется снова.

Кучей в плавной сортировке считается следующая структура, состоящая из 3 элементов:

1. Порядок кучи — целое неотрицательное число;
2. Корень кучи — один элемент массива;
3. Дочерние кучи (подкучи): если порядок кучи d больше одного, куча имеет две дочерние кучи с порядками $(d-1)$ и $(d-2)$, иначе куча дочерних куч не имеет и состоит только из корня;

Таким образом куча содержит $L(d)$ элементов массива, где $L(d)$ — d-тое число Леонарда:

$$L(x) = L(x-1) + L(x-2) + 1, \quad L(0) = 1,\ L(1) = 1$$

$$L(x) = 1,\ 1,\ 3,\ 5,\ 9,\ 15,\ 25\ldots$$

Для дочерних куч должно выполняться следующее свойство: корень родительской кучи должен быть не меньше корня дочерней кучи.

Часть массива сформирована в кучу порядка d, если сначала идет часть массива, сформированная в кучу порядка $(d-1)$, потом часть массива, сформированная в кучу порядка $(d-2)$, потом идет корень кучи, который не меньше корней подкуч. Таким образом последним элементом части массива, сформированной в кучу, будет максимальный элемент. Размер такой части должен быть одним из чисел Леонарда.

Массив произвольной длины можно преобразовать в последовательность куч, для которой будут выполняться следующие свойства:

1. Корень каждой следующей кучи не меньше корня предыдущей;
2. Порядки куч строго убывают, притом все кучи кроме последней не могут иметь порядок, отличающийся на единицу от порядка предыдущей кучи.

В массиве, представляющем такую последовательность куч, последний элемент будет наибольшим.

Сортировка состоит из двух этапов:

1. Добавление по одному элементу к имеющимся кучам, пока весь массив не будет сформирован в последовательность куч;

2. Удаление по одному элементу из куч.

После каждой итерации необходимо восстанавливать свойства куч переставлением элементов массива.

Добавления одного элемента в последовательность куч происходит по следующему правилу:

1. Если последние две кучи имеют порядки d и $(d-1)$, то новый элемент становится корнем новой кучи порядка $(d+1)$, для которой последние две кучи будут подкучами.

2. Иначе добавляется новая куча, состоящая из одного корня. Если последняя куча в последовательности имеет порядок 1, то новая куча будет иметь порядок 0, иначе 1.

После этого для новой кучи вызывается процедура восстановления свойств последовательности куч. Новая куча становится текущей. Восстановление происходит в два этапа:

1. Перемещение: если корень следующей кучи больше корня текущей кучи и наибольшего из корней подкуч текущей кучи, то корень текущей кучи меняется местами с корнем следующей; следующая куча становится текущей. Когда перемещение больше невозможно, начинается следующая стадия.

2. Просеивание: если порядок кучи больше 1 и корень текущей кучи меньше максимального корня своих подкуч, меняем корни местами; эта подкуча становится текущей кучей.

Когда все элементы исходного массива были добавлены в последовательность куч, элементы начинают удаляться по одному по следующему правилу:

1. Если последняя куча в последовательности имеет порядок 1 или 0, то эта куча удаляется. После этого удаления все свойтсва будут все еще выполнены.

2. Иначе из последней кучи удаляется корень, в последовательность добавляются ее подкучи. Для левой и правой подкучи надо вызвать процедуру восстановления свойств последовательности куч.

Когда все элементы будут удалены из последовательности, массив будет отсортирован.

В худшем случае алгоритм имеет сложность $O(N \log N)$, но если данные частично отсортированы, то скорость приближается к $O(N)$. В этом заключается преимущество перед обычной пирамидальной сортировкой. Также процесс просеивания требует меньше проверок, так как наличие дочерних куч строго определяется порядком кучи.

1 Постановка задачи

В рамках производственной практики необходимо разработать специализированный процессор для сортировки массива алгоритмом плавной сортировки. Размер массива составляет от 2 до 40 элементов, массив состоит из двоичных беззнаковых чисел разрядностью 16 бит. Для процессора необходимо разработать набор машинных команд и программу, реализующую алгоритм сортировки.

Также необходимо разработать подсистему вывода отсортированного массива на дисплей.

При разработке процессора необходимо использовать исполнительные устройства (регистры общего назначения и арифметико-логические устройства) разрядностью 8 бит.

Процессор должен быть разработан в программе «Logisim».

2 Метод решения

2.1 Структура данных

Для реализации алгоритма плавной сортировки необходимы следующие структуры данных:

1. Сортируемый массив: последовательность значений разрядностью 16 бит и длина последовательности.

2. Стек куч: последовательность целых неотрицательных чисел, порядки куч в последовательности куч.

3. Количество обработанных элементов массива: целое неотрицательное число, не превосходит размер сортируемого массива. Хранение этого числа является необязательным, так как оно может быть вычислено на основании стека куч.

4. Локальная данные для процедуры восстановления свойств последовательности куч: номер текущей кучи, адрес корня текущей кучи, порядок текущей кучи.

2.2 Архитектура компьютера

Процессор разработан с использованием гарвардской архитектуры. Разделение памяти для операндов и памяти для инструкций на два различных элемента было сделано, потому что размещение программы в памяти заняло бы большое количество восьмибитных ячеек (порядка 256). Из-за этого адрес пришлось бы сделать разрядностью 9 бит, а такой адрес не помещается в регистр разрядностью 8 бит.

Процессор состоит из регистров общего назначения, счетчика команд, памяти для операндов, памяти для команд, арифметико-логического устройства, подсистемы вывода на дисплей, таблицы чисел Леонарда, декодера, подсистемы условных переходов, шина данных.

Общая структура процессора представлена на рис. 1.

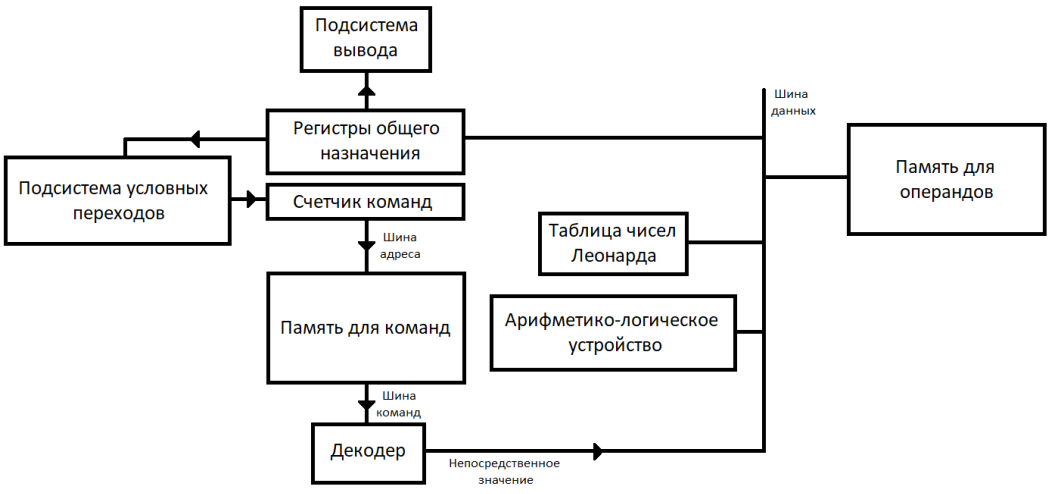

Рис. 1: Общая структура процессора

2.2.1 Память для операндов

Память для операндов имеет разрядность данных 8 бит, разрядность адреса 7 бит. Адрес перед чтением или записью записывается в адресный регистр. Регистр имеет разрядность 7 бит и является счетчиком для того чтобы можно было получить доступ к двум последовательным ячейкам памяти без дополнительных вычислений адреса. Адрес загружается в адресный регистр с шины данных.

При обращении на чтение или запись обмен данными происходит с шиной данных.

2.2.2 Память для команд

Память для команд имеет разрядность данных 16 бит и разрядность адреса 7 бит. Память работает только на чтение, данные после чтения поступают в декодер. Адрес поступает из счетчика команд.

2.2.3 Регистры общего назначения

Процессор имеет 9 регистров общего назначения. Общая информация о регистрах представлена в таблице 1.

Таблица 1: Регистры общего назначения

Код	**Название**	**Разрядность**	**Счетчик**	**Значение по умолчанию**
0000	RA	8	нет	0x00
0001	RB	8	нет	0x00
0010	RC	8	нет	0x00
0011	RD	8	нет	0x00
0100	AR1	7	нет	0x00
0101	AR2	7	нет	0x00
0110	ASR	7	нет	0x00
0111	HSP	7	да	0x00
1000	AP	7	да	0x00

Код регистра используется для мультиплексирования, он указывается в команде, если аргументом является регистр.

Регистры **RA-RD** используются для хранения операндов, загруженных из памяти.

Регистры **AR1**, **AR2**, **ASR** используются для адресации широкого операнда в памяти. При мультиплексировании регистра с кодом 1110 на шину поступает значение $(ASR - AR1) \cdot 2$. При мультиплексировании регистра с кодом 1111 на шину поступает значение $(ASR - AR2) \cdot 2$. Регистры имеют разрядность 7 бит, так как используются для адресации памяти с разрядностью адреса 7 бит.

Регистр **HSP** хранит адрес следующего элемента за последним элементом стека куч. Регистр имеет разрядность 7 бит, так как используется для адресации памяти с разрядностью адреса 7 бит. Регистр является счетчиком, так как операции помещения значения в стек куч и удаления значения из стека куч требуют изменения значения регистра на 1.

Регистр **AP** хранит количество обработанных элементов массива. Регистр имеет разрядность 7, так как это значение используется как адрес в памяти. Регистр является счетчиком,

так как в алгоритме часто используется увеличения количества элементов обрабатываемого массива на 1.

При обращении к регистрам на чтение и запись данные поступают с шины данных.

2.2.4 Счетчик команд

Счетчик команд имеет разрядность 7 бит. Счетчик команд имеет код, как и регистры общего назначения, но использование счетчика команд в качестве аргумента команды приведет к ошибке. Счетчик команд имеет код 1001.

2.2.5 Арифметико-логическое устройство

Арифметическо-логическое устройство состоит из регистра левого операнда и сумматора. Регистр имеет разрядность 8 бит. С помощью арифметико-логического устройства возможно выполнять операции сложения и вычитания.

Операнды в арифметико-логическое устройство поступает с шины данных. Результат также поступает на шину данных.

2.2.6 Таблица чисел Леонардо

Таблица чисел Леонардо является постоянным запоминающим устройством с разрядностью данных 8 бит и разрядностью адреса 3 бита. Устройство хранит последовательно первые 8 чисел Леонардо. Обмен данными происходит с шины данных.

2.2.7 Подсистема условных переходов

Подсистема условных переходов реализует логику безусловного перехода и три варианта условного перехода. Они подробнее описаны в разделе 2.3.

2.2.8 Подсистема вывода

Подсистема вывода состоит из 5 шестнадцатеричных индикаторов, двух регистров данных и триггера вывода. Регистры данных имеют разрядность 8 бит и получают на вход значения регистров общего назначения **RA**, **RB**. Также система имеет 5 делителей, которые переводят значение в регистрах данных в 5 десятичных цифр. Если в триггере вывода хранится 0, на индикаторах отобразятся черточки, иначе отобразится десятичное число, соответствующее данным в регистрах.

2.2.9 Декодер

Декодер имеет двухступенчатую структуру. На первом этапе определяется группа команд, к которой относится команда. На втором этапе задействуются необходимые управляющие входы. Для групп команд, выполнение которых требует нескольких тактов, используется счетчик шагов и lookup table.

2.3 Набор команд

Для реализации алгоритма плавной сортировки был разработан следующий набор команд:

1. **MOV** ***SRC DST***. Загружает в регистр ***DST*** либо значение из регистра ***SRC***, либо непосредственное значение ***SRC***. Непосредственное значение имеет разрядность 7 бит.

2. **L** ***SRC DST***. Загружает в регистр ***DST*** число Леонарда $L(x)$, где x — значение регистра ***SRC***.

3. **LD** ***SRC DST***. Загружает в регистр ***DST*** значение из памяти для операндов либо по адресу в регистре ***SRC***, либо по непосредственному адресу ***SRC***. Непосредственное значение имеет разрядность 7 бит.

4. **ST** ***DST SRC***. Сохраняет значение регистра ***SRC*** в память для операндов либо по адресу в регистре ***DST***, либо по непосредственному адресу ***DST***. Непосредственное значение имеет разрядность 7 бит.

5. **HPUSH** ***SRC***. Добавляет в стек куч значение регистра ***SRC***. Также увеличивается значение регистра **HSP**.

6. **HPOP** ***DST***. Удаляет из стека куч последний элемент и сохраняет в регистр ***DST***. Также уменьшается значение регистра **HSP**.

7. **ADD** ***SRC RT DST***. Складывает значение в регистре ***SRC*** либо со значением регистра ***RT***, либо с непосредственным значением ***RT***. Результат сохраняется в регистр ***DST***. Непосредственное значение имеет разрядность 3 бита.

8. **SUB** ***SRC RT DST***. Вычитает из значения в регистре ***SRC*** либо значение регистра ***RT***, либо непосредственное значение ***RT***. Результат сохраняется в регистр ***DST***. Непосредственное значение имеет разрядность 3 бита.

9. **LDAW** ***DST SRC***. Загружает два последовательных байта из памяти для операндов по адресу в регистре ***SRC*** в два регистра общего назначения. Если ***DST***$= 0$, то в регистры **RA**, **RB**, иначе в регистры **RC**, **RD**. В регистрах **RD**, **RB** хранятся старшие части.

10. **STAW** ***SRC DST***. Сохраняет последовательно два байта из двух регистров общего назначения в память для операндов, начиная с адреса в регистре ***DST***. Если ***SRC***$= 0$, то из регистров **RA**, **RB**, иначе из регистров **RC**, **RD**. В регистрах **RD**, **RB** хранятся старшие части.

11. **STEP** ***CNT DIR***. Увеличивает или уменьшает значение счетчика. Если ***CNT***$= 0$, то изменяется значение указателя на конец стека куч, иначе — значение указателя на последний элемент обрабатываемой части массива.

12. **B** ***ADR***. Безусловный переход на команду по адресу ***ADR***. Адрес имеет разрядность 7 бит.

13. **BL2** ***ADR***. Переход на команду по адресу ***ADR***, если в регистре **RA** значение меньше 2. Адрес имеет разрядность 7 бит.

14. **BEOA** ***ADR***. Переход на команду по адресу ***ADR***, если значение регистра **AP** равно количеству элементов в массиве, то есть достигнут конец массива. Адрес имеет разрядность 7 бит.

15. **BCMP *ADR***. Переход на команду по адресу ***ADR***, если в широкий операнд в регистрах **RA**, **RB** больше широкого операнда в регистрах **RC**, **RD**, причем в регистрах **RD**, **RB** хранятся старшие части. Адрес имеет разрядность 7 бит.

16. **LINK *ADR***. Сохраняет непосредственное значение ***ADR*** в регистр для адреса возврата. Непосредственное значение имеет разрядность 7 бит.

17. **RET**. Загружает значение из регистра адреса возврата в счетчик команд.

18. **OUT *F***. Загружает значение из регистров **RA**, **RB** в регистры вывода на дисплей. Если ***F***= 0, то на дисплее будут черточки, иначе на дисплее будет десятичное число, соответствующее значению в регистрах вывода.

19. **HLT**. Выключает тактирование.

20. **NOP**. Загружает команду по адресу из счетчика команд.

Машинные коды команд представлены на рис. 2. Бит **I** равен 1, если в качестве операнда используется непосредственное значение, иначе бит равен 0.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NOP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RET	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0
HLT	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
OUT F	0	0	0	0	1	0	0	0	0	0	0	F	0	0	0	0
MOV SRC DST	1	0	0	0	I	SRC							DST			
L SRC DST	1	0	0	1	0	0	0	0	SRC				DST			
LD SRC DST	1	1	1	0	I	SRC							DST			
ST DST SRC	1	1	0	0	I	DST							SRC			
HPUSH SRC	1	1	0	1	0	0	0	0	0	1	1	1	SRC			
HPOP DST	1	1	1	1	0	0	0	0	0	1	1	1	DST			
LDAW R SRC	0	1	1	0	0	0	0	0	0	0	0	R	DST			
STAW R DST	0	1	0	0	0	0	0	0	0	0	0	R	SRC			
STEP C D	0	1	D	1	0	0	0	0	0	0	0	C	0	0	0	0
ADD SRC RT DST	1	0	1	0	I	SRC				RT			DST			
SUB SRC RT DST	1	0	1	1	I	SRC				RT			DST			
B ADR	0	0	1	0	1	ADR							0	0	0	0
BL2 ADR	0	0	1	0	1	ADR							0	0	1	0
BEOA ADR	0	0	1	0	1	ADR							0	0	0	1
BCMP ADR	0	0	1	0	1	ADR							0	0	1	1
LINK ADR	0	0	1	1	1	ADR							0	0	0	0

Рис. 2: Машинные коды команд

Команды имеют продолжительность от 1 до 3 тактов. Продолжительности команд представлены в таблице 2.

Таблица 2: Продолжительность выполнения команд

Команда	**Продолжительность выполнения (тактов)**
NOP	1
HLT	1
OUT	1
RET	2
MOV	1
L	1
LD	2
ST	2
HPUSH	2
HPOP	2
LDAW	3
STAW	3
STEP	1
ADD	2
SUB	2
B	1
BL2	1
BEOA	1
BCMP	1
LINK	1

2.4 Симулятор процессора

Для отладки программы, реализующей алгоритм, был разработан симулятор разрабатываемого процессора. Симулятор представляет из себя графическое приложение Windows. Симулятор содержит программные представления памяти для операндов, регистров общего назначения, счетчика команд и разработанного набора команд. Логика выполнения команд реализована на языке высокого уровня. В симуляторе есть возможность написать программу, состоящую из команд из реализованного набора команд, выполнить эту программу. В симуляторе реализованы выполнение по шагам и точки останова. В симуляторе есть возможность протестировать программу на большом количестве массивов с автоматической проверкой и сбором статистики. Также при помощи симулятора можно сгенерировать образ памяти для команд с программой для последующей загрузки в Logisim. Для удобства программирования есть возможность заменять адреса команд мнемоническими метками, при выполнении программы или генерации кода программа подставит вместо них необходимые адреса. Также симулятор выдает сообщения об ошибках и неразрешенных метках.

3 Решение поставленной задачи

3.1 Написанная программа

Программа, реализующая алгоритм плавной сортировки на разработанном процессоре, была написана в двух версиях. Первая версия не содержит бесконечного цикла вывода отсортированного массива на дисплей, использовалась в автоматических тестированиях средствами разработанного симулятора. Программа содержит 118 команд. Вторая версия содержит бесконечный цикл для вывода на дисплей. Программа содержит 128 команд.

Текст второй программы приведен в приложении А.

3.2 Схемотехническое решение

3.2.1 Шина данных и арифметико-логическое устройство

Для ускорения работы команд, использующих таблицу чисел Леонарда и арифметико-логическое устройство было использована структура, представленная на рис. 3. Таким образом на шину данных может попасть либо значение одного из регистров, либо число Леонарда, либо значение выхода арифметико-логического устройства. Чтобы на шину данных попало непосредственное значение из декодера, значение регистра арифметико-логического устройства сбрасывается и на шину данных пропускается выход арифметико-логического устройства. Схемотехническая реализация этой части представлена на рис. 4.

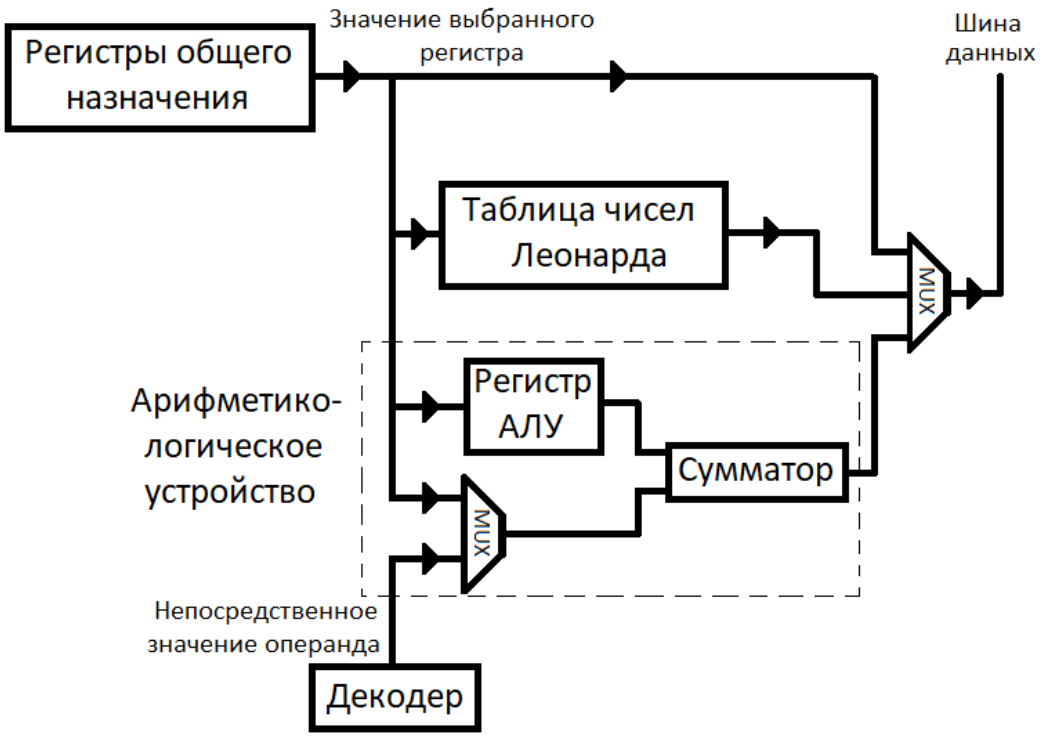

Рис. 3: Блок-схема реализации арифметико-логического устройства и шины данных

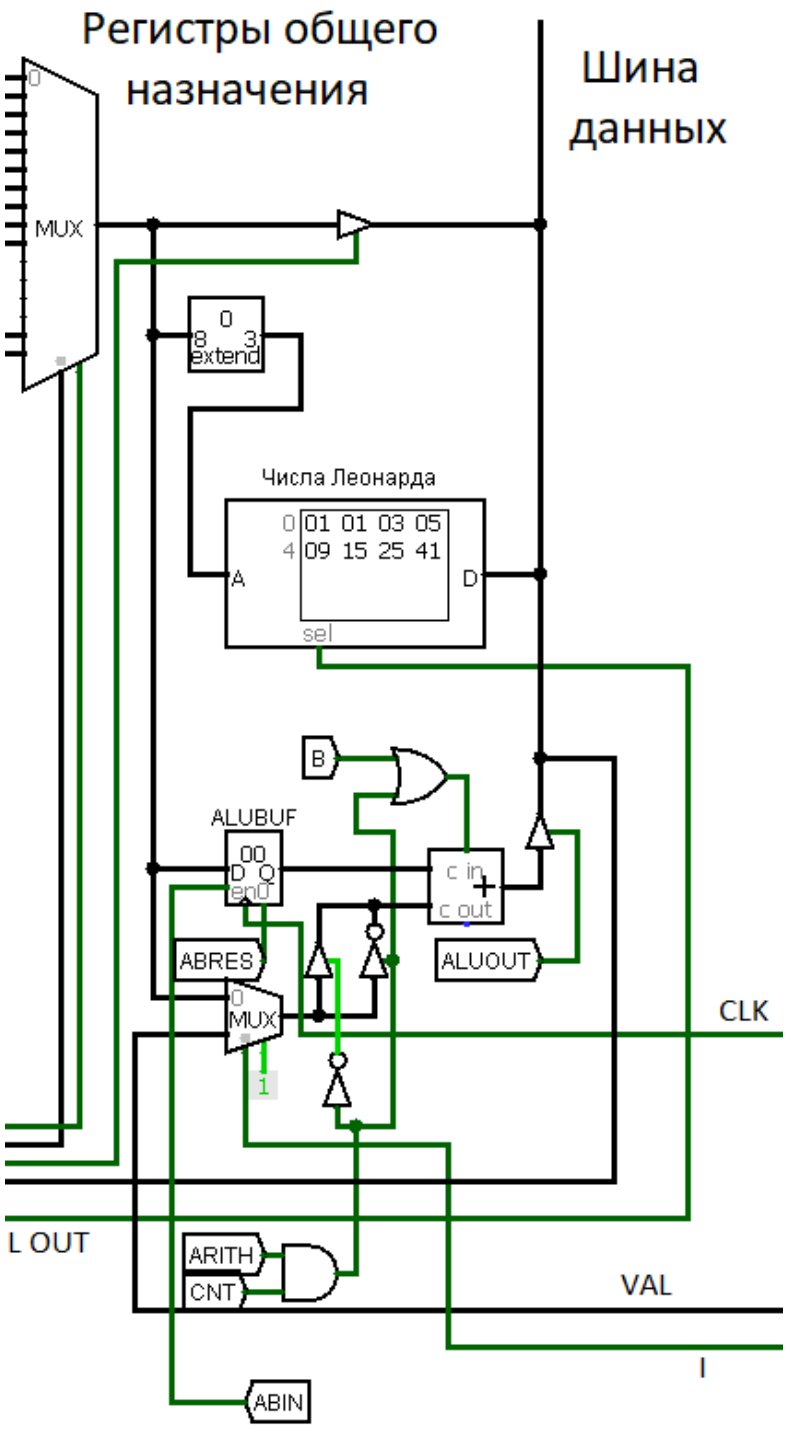

Рис. 4: Схемотехническая реализация арифметико-логического устройства и шины данных

3.2.2 Память для операндов

Память для операндов организована следующим образом:

- Массив располагается по адресам 0x00-0x4f. Младшая часть широкого операнда имеет младший адрес.

- Стек куч располагается по адресам 0x50-0x57.
- По адресу 0x70 находится размер массива.

Адресный регистр является счетчиком. При работе с широкими операндами в него загружается адрес младшей части, после обращения к ней значение в адресном регистре увеличивается на 1.

Для ускорения работы команды **HPOP** между шиной данных и входом адресных регистров стоит вычитатель. При выполнении команды **HPOP** из поступающего адреса вычитается 1, при выполнении остальных команд значение поступает в адресный регистр без изменений.

Схемотехническая реализация памяти для операндов представлена на рис. 5.

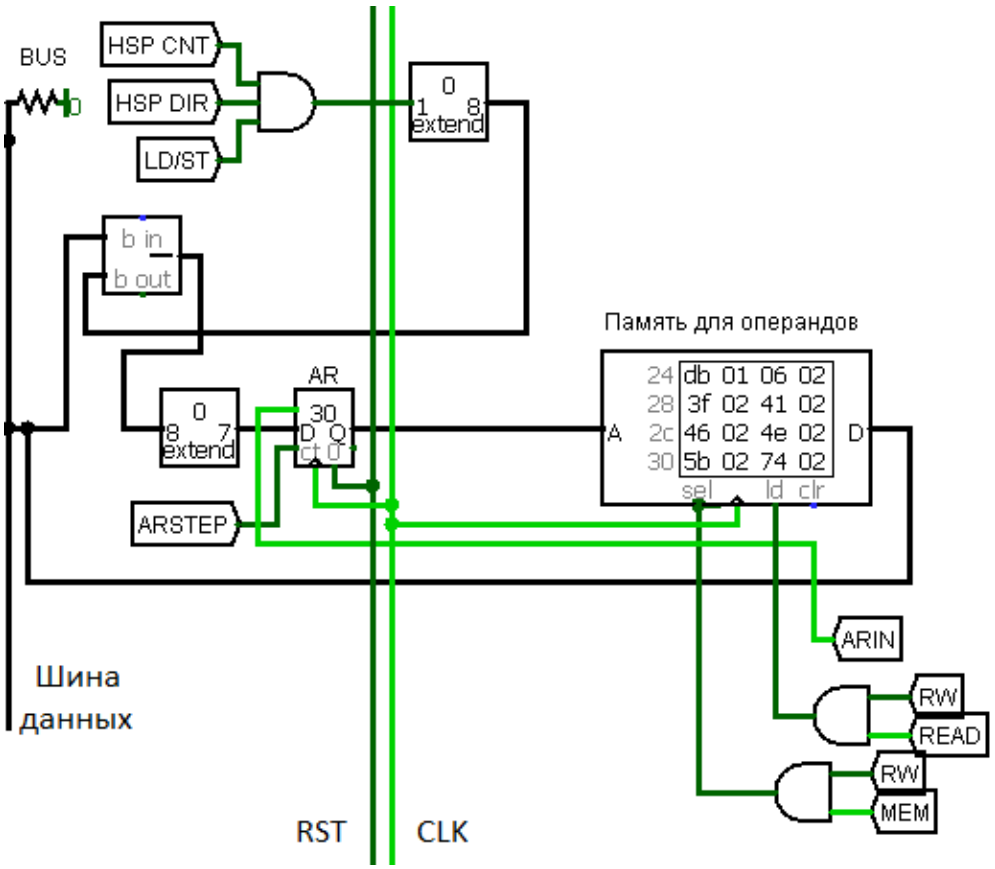

Рис. 5: Схемотехническая реализация памяти для операндов

3.2.3 Регистры общего назначения и счетчик команд

Входы данных регистров подключены к шине данных, управляющие вход регистров подключены через демультиплексор. Выходы данных регистров через мультиплексор пускаются на внутренний контур (см. раздел 3.2.1), откуда значение может быть отправлено на шину данных.

Для реализации логики адресации со смещением при помощи регистров **AR1**, **AR2** и **ASR** (см. раздел 2.2.3), к последним входам мультиплексора подключен выход вычитателя.

Выходы регистров **RA**, **RB** подключены напрямую к входам регистров данных подсистемы вывода на дисплей. Выходы регистров **RA**-**RC**, **AP** подключены напрямую к подсистеме условных переходов.

Выход счетчика команд (регистр **PC**) подключен непосредственно к памяти для операндов.

Также к мультиплексору подключен регистр **CR** из подсистемы условных переходов, что позволяет загружать размер массива из памяти при помощи команд загрузки в регистр. Этот регистр имеет код 1010.

Схемотехническая реализация регистров общего назначения представлена на рис. 6.

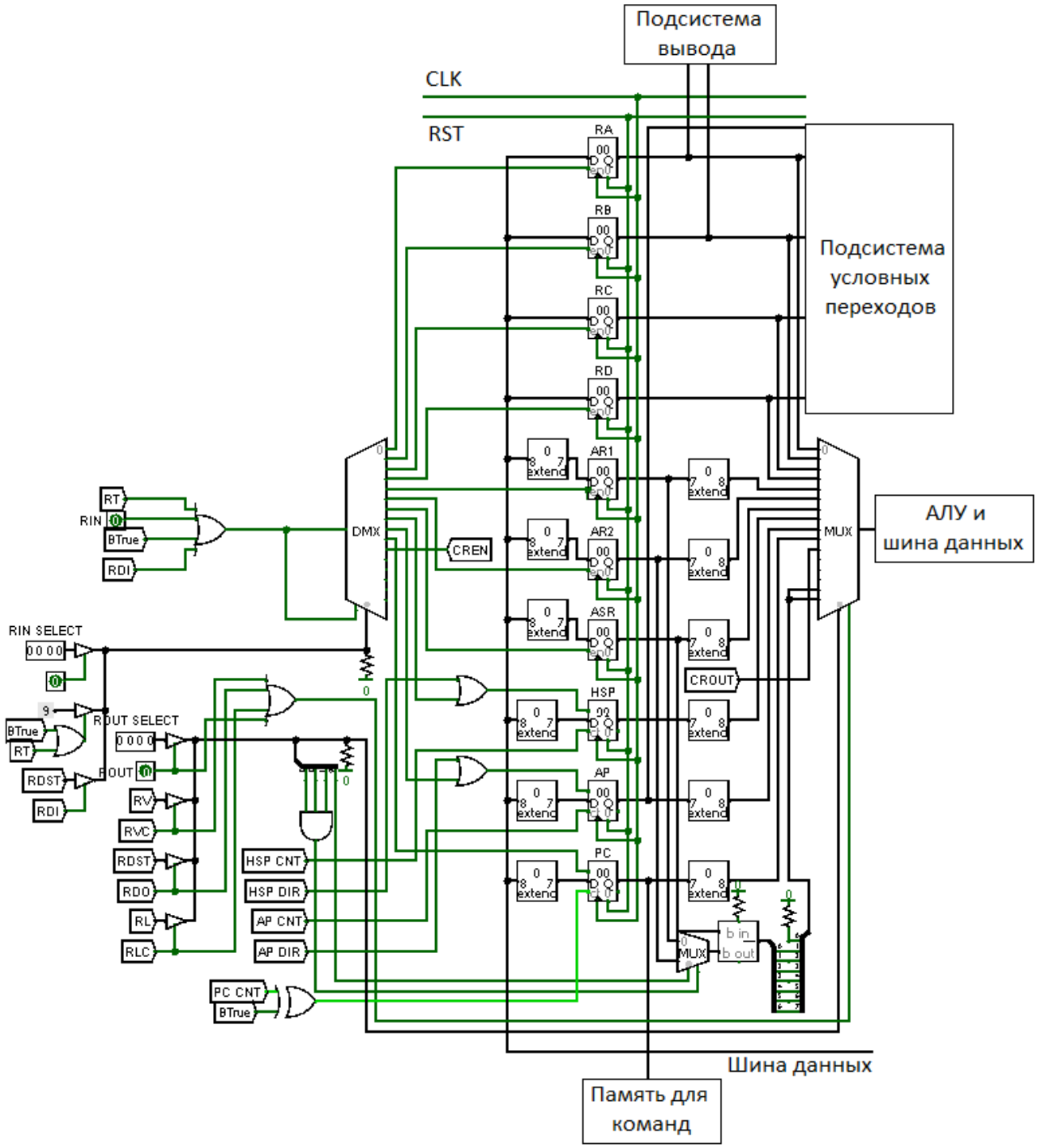

Рис. 6: Схемотехническая реализация регистров общего назначения

3.2.4 Подсистема условных переходов

Подсистема условных переходов состоит из нескольких частей. Первая часть определяет, выполняется ли условие для перехода. Вторая часть загружает необходимые значения в счетчик команд и командный регистр декодера.

Подсистема поддерживает три типа условных переходов и безусловный переход. Типы

условных переходов:

1. Значение в регистре **RA** меньше двух. Проверка этого условия осуществляется одним элементом ИЛИ-НЕ, подключенным к 7 старшим битам регистра **RA**.

2. Широкое значение в регистрах **RA**, **RB** больше, чем широкое значение в регистрах **RC**, **RD**. Реализовано с помощью двух компараторов разрядностью 8 бит.

3. Значение регистра **AP** не меньше размера массива, хранящегося в регистре **CR**. Сравнение реализовано один компаратором разрядностью 8 бит.

Сигнал от необходимого условия или постоянное значение 1 (безусловный переход) выбирается мультиплексором и отправляется во вторую часть подсистемы условных переходов.

Схемотехническая реализация первой части подсистемы условных переходов представлена на рис. 7.

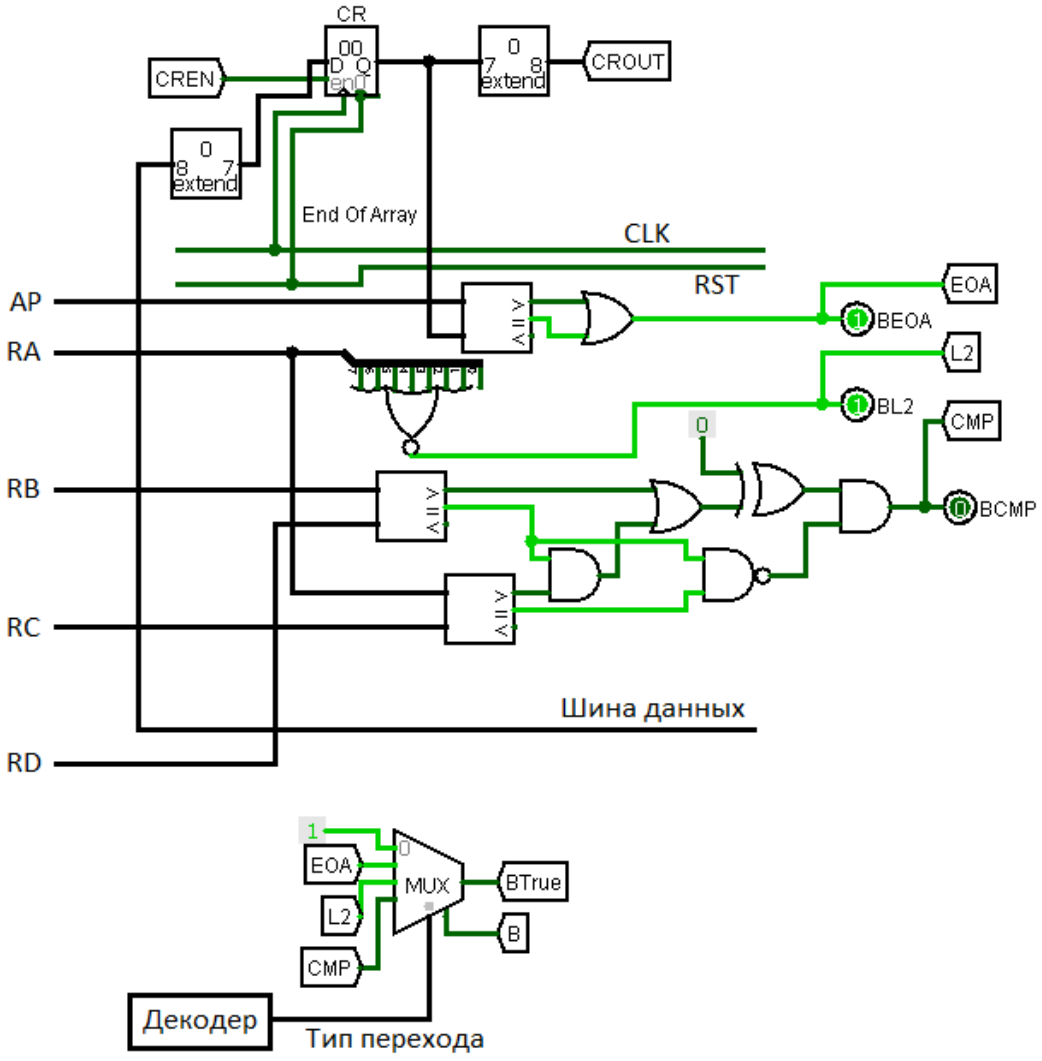

Рис. 7: Реализация первой части подсистемы условных переходов

Если необходимо сделать переход, адрес необходимой команды подается непосредственно в адресный вход памяти команд, а увеличенное на 1 значение загружается в счетчик команд. Для этого обнуляется значение регистра арифметико-логического устройства, на вход передается адрес, а на вход переноса сумматора подается 1. Результат операции оказывается на шине данных, оттуда загружается в счетчик команд. Если переход не нужно делать, на адресный вход памяти команд передается значение счетчика команд, а для счетчика команд устанавливается режим счета. В обоих случаях в регистры декодера загружается значение следующей команды из памяти команд. Схемотехническая реализация второй части присутствует на рис. 8, 6, 4.

3.2.5 Память для команд

Схемотехническая реализация памяти для команд представлена на рис. 8.

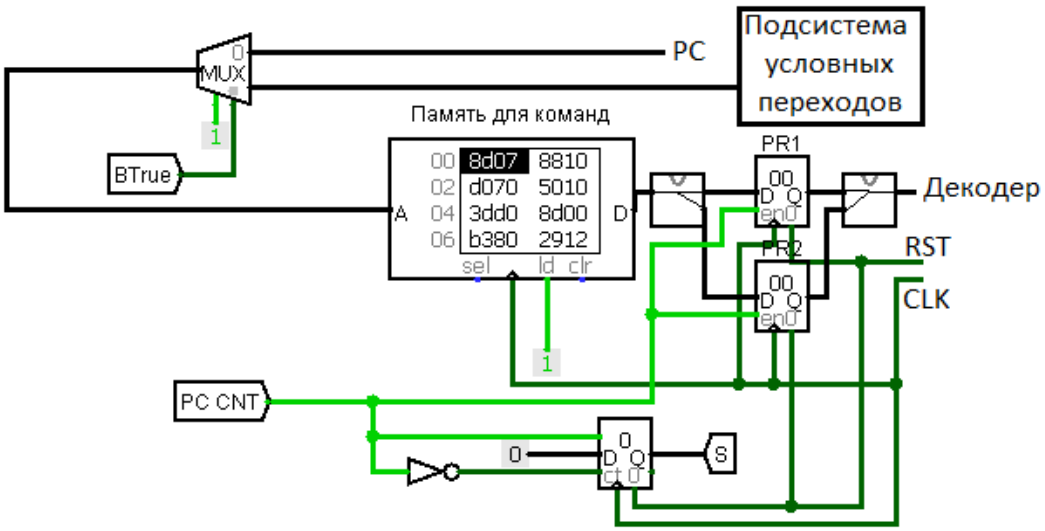

Рис. 8: Схемотехническая реализация памяти для команд

3.2.6 Декодер

Декодер имеет сложную структуру и состоит из нескольких частей:

1. Декодер группы команд. Некоторые команды имеют схожую логику выполнения, различия заключаются в нескольких управляющих входах.

2. Lookup table. Для групп команд, выполнение которых требует более 1 такта использовано постоянное запоминающее устройство, адресом в котором является код группы команд и значение счетчика шагов, а значением является значения подмножетства управляющих входов. Счетчик шагов изображен на рис. 8.

3. Система обработки аргументов. Из кода команды разрядностью 16 бит извлекаются значения аргументов согласно спецификации команд (см. рис. 2).

4. Система управления управляющими входами.

На последнем такте каждой команды увеличивается значение счетчика команд и значение следующей команды записывается в регистры декодера. Все команды разделяется на следующие группы:

- Перемещение в регистр. Команды: **MOV**, **L**. Длина выполнения: 1 такт.
  Код команд этой группы начинается с 100x. В зависимости от значения x на шину данных попадает значение либо с выхода арифметико-логического устройства, либо из таблицы чисел Леонарда. Значение с шины данных загружается в регистр.

- Загрузка / сохранение значений. Команды: **LD**, **ST**, **HPUSH**, **HPOP**. Длина выполнения: 2 такта.
  Код команд этой группы начинается с 11xy. На первом такте на шину данных попадает значение адреса и загружается в адресный регистр памяти для операндов. На втором, если x = 0, происходит обращение к памяти для операндов на запись и на шину данных поступает значение из выбранного регистра; если x = 1, происходит обращение к памяти для операндов на чтение, прочитанное значение записывается в выбранный регистр. Если y = 1, происходит изменение счетчика **HSP**, в большую сторону, если x = 0, в меньшую, если x = 1.

- Арифметические операции. Команды: **ADD**, **SUB**. Длина выполнения: 2 такта.
  Код команд этой группы начинается с 101x. На первом такте значение выбранного регистра загружается в регистр арифметико-логического устройства. На втором такте значение с выхода арифметико-логического устройства поступает на шину данных и записывается в выбранный регистр. Если x = 0, устройство настраивается на сложение, иначе на вычитание.

- Работа с широкими операндами. Команды: **LDAW**, **STAW**. Длина выполнения: 3 такта.
  Код команд этой группы начинается с 01x0. На первом такте на шину данных значение выбранного регистра и записывается в адресный регистр. На втором и третьем такте происходит обращение к памяти для операндов на запись, если x = 0, или на чтение, если x = 1. На втором такте значение адресного регистра увеличивается на 1.

- Команда **STEP**. Длина выполнения: 1 такт.
  Код команд этой группы начинается с 01x1. Изменяет при помощи системы управления счетчиками значение счетчика в указанную сторону.

- Условные переходы. **B**, **BL2**, **BCMP**, **BEOA**. Длина выполнения: 1 такт.
  Код команд этой группы начинается с 00101. Логика работы этих команд подробно описана в разделе 3.2.4.

- Команда **LINK**. Длина выполнения: 1 такт.
  Код команд этой группы начинается с 00111. Значение операнда записывается в регистр адреса возврата **RET**.

- Команда **RET**. Длина выполнения: 2 такта.
  Код команд этой группы начинается с 00011. На первом такте из регистра адреса возврата **RET** на шину данных поступает значение и записывается в счетчик команд. На втором такте в регистры декодера записывается значение необходимой команды и увеличивается значение счетчика команд.

- Команда **OUT**. Длина выполнения: 1 такт.
  Код команд этой группы начинается с 00001. В регистры подсистемы вывода загружаются значения регистров **RA**, **RB**, в триггер вывода записывается переданное значение.

- Команда **HLT**. Длина выполнения: 1 такт.
  Код команд этой группы начинается с 00010. Команда выключает тактирование.

- Команда **NOP**. Длина выполнения: 1 такт.
  Код команд этой группы начинается с 00000. Команда записывает в регистры декодера значение необходимой команды и увеличивает значение счетчика команд.

Схемотехническая реализация определителя группы команд представлена на рис. 9.

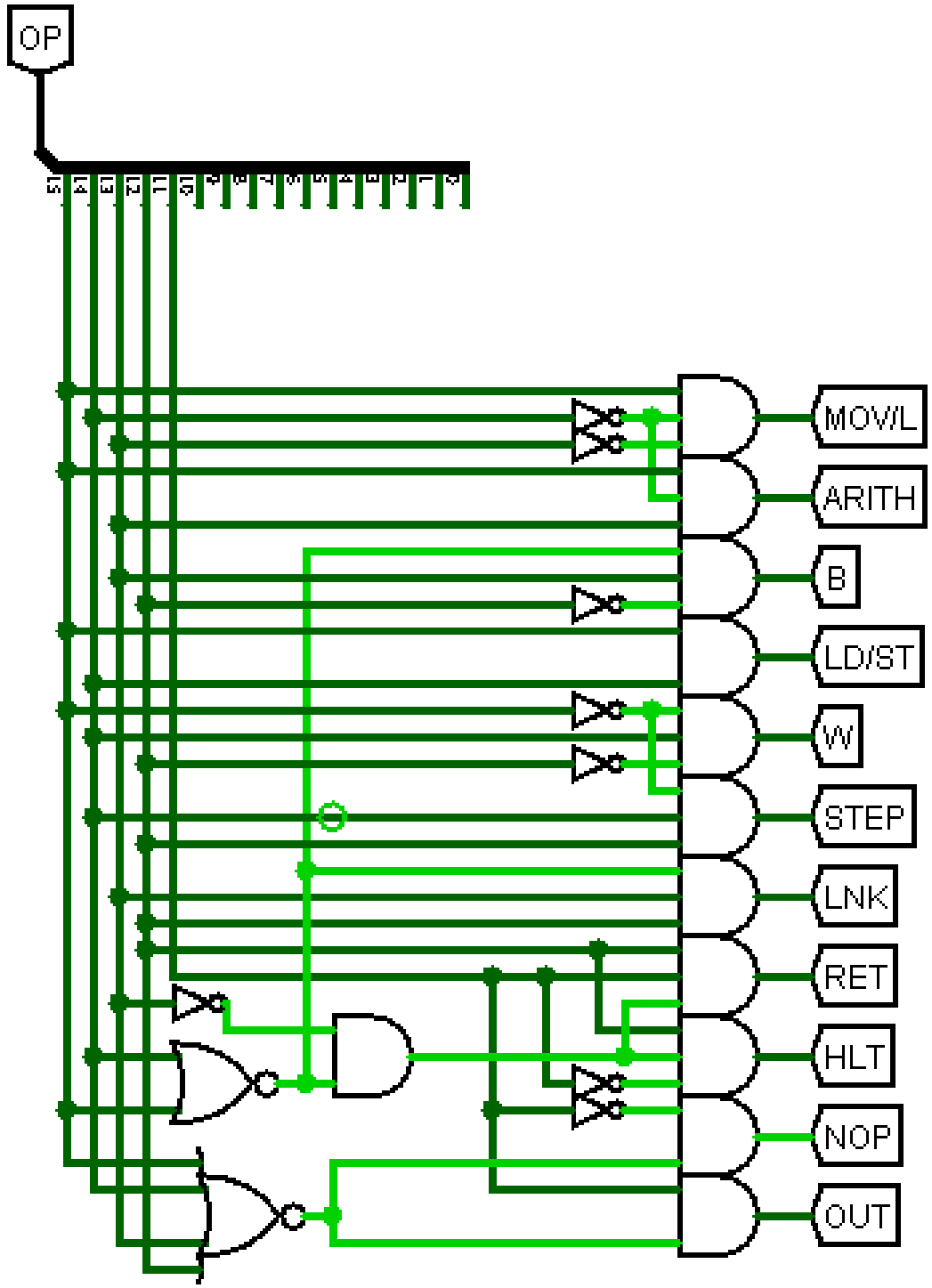

Рис. 9: Схемотехническая реализация определителя группы команд

Схемотехническая реализация системы обработки аргументов представлена на рис. 10.

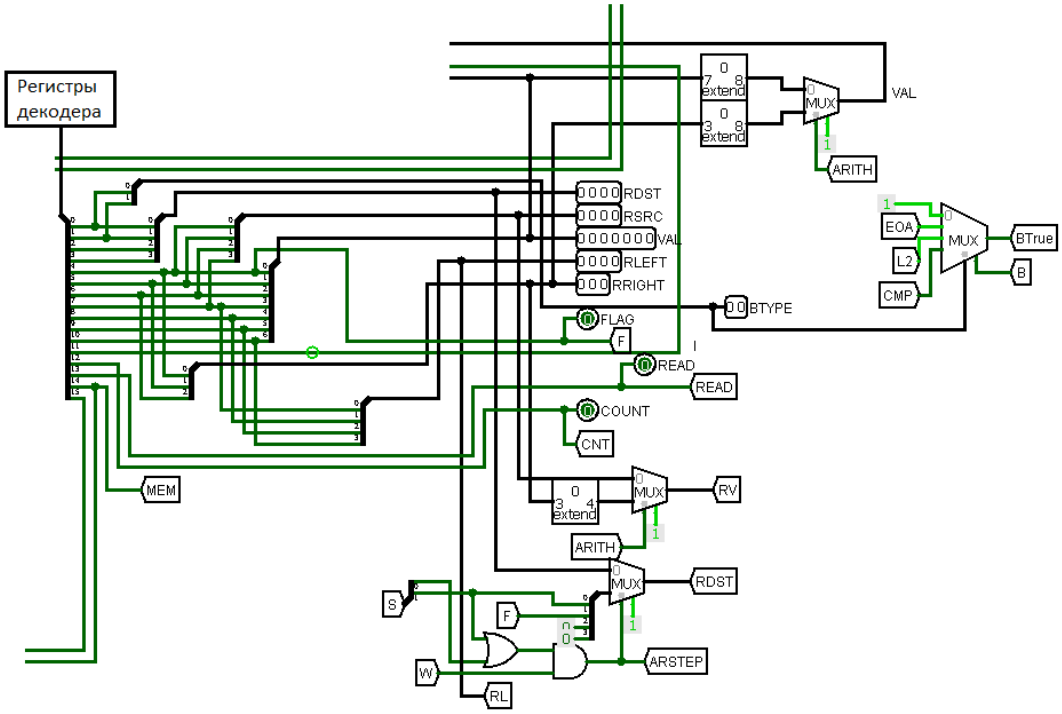

Рис. 10: Схемотехническая реализация системы обработки аргументов

Схемотехническая реализация lookup table и подключения управляющих входов представлена на рис. 11. В приложении Б представлен образ памяти для lookup table.

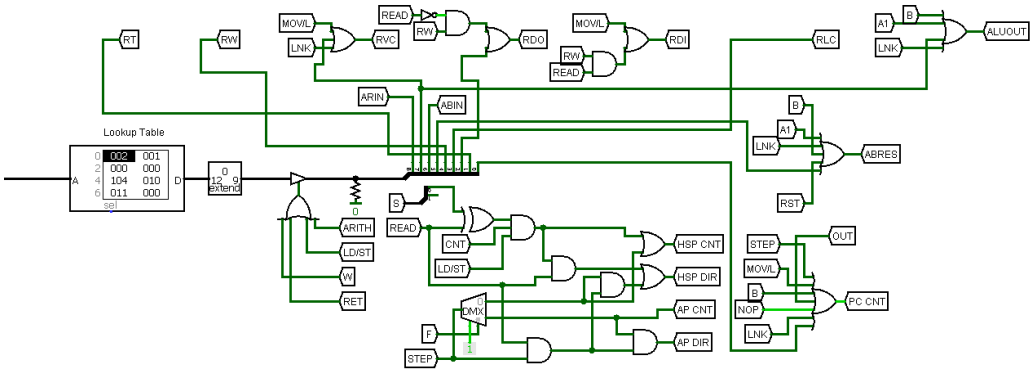

Рис. 11: Схемотехническая реализация lookup table и подключения управляющих входов

3.2.7 Подсистема вывода

Схемотехническая реализация подсистемы вывода представлена на рис. 12.

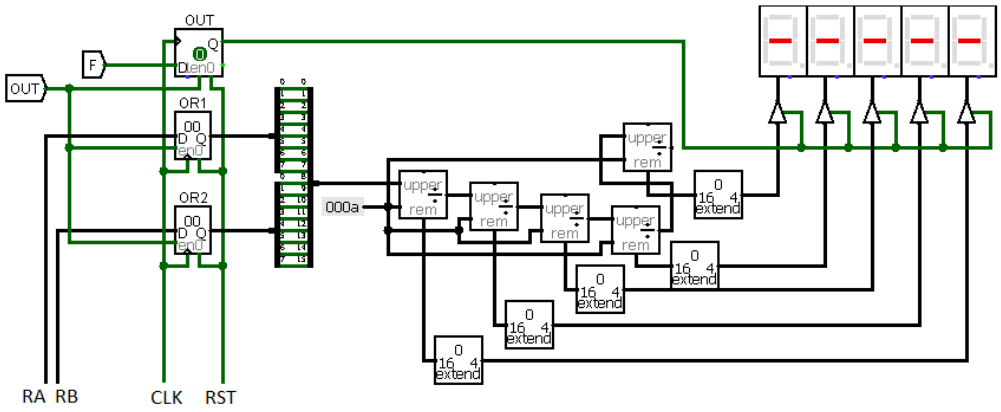

Рис. 12: Схемотехническая реализация подсистемы вывода

3.2.8 Сохранения адреса возврата

Для сохранения адреса возврата используется отдельный регистр **RET**. Регистр имеет разрядность 7 бит.

Запись в регистр происходит командой **LINK**, в регистр записывается непосредственное значение из команды. Обращение на чтение происходит при использовании команды **RET**. Значение из регистра попадает в счетчик команд.

Схемотехническая реализация регистра адреса возврата представлена на рис. 13.

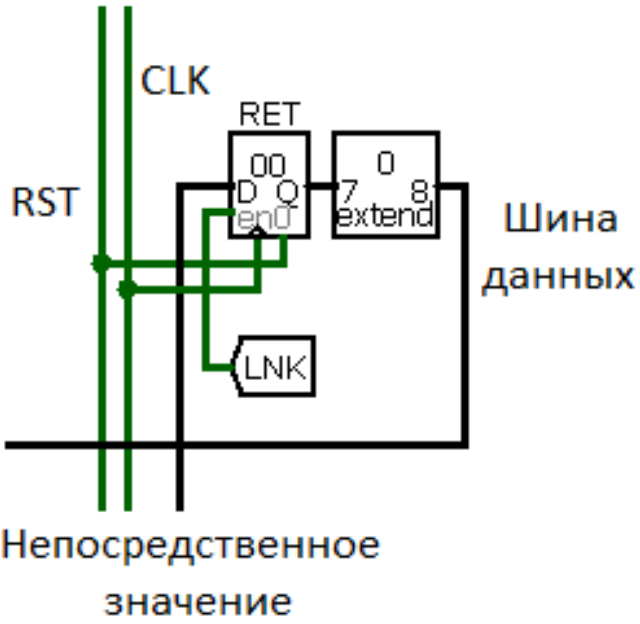

Рис. 13: Схемотехническая реализация регистра адреса возврата

4 Результат работы

В рамках производственной практики был разработан специальный процессор для сортировки массива плавной сортировкой.

В программе Logisim была протестирована работа процессора на последовательностях разной длины. Результаты тестирования приведены в таблице 3.

Таблица 3: Результаты тестирования

Длина последовательности	**Количество инверсий**	**Успешно**	**Количество тактов**
6	10	да	730
18	88	да	3900
35	296	да	9785

Средствами разработанного симулятора были проведено тестирование программы на 50 массивах разной длины, все массивы отсортированы успешно.

Список литературы

[1] ARMv7-M Architecture Application Level Reference Manual, 2008.

[2] Курс лекций по архитектуре суперкомпьютеров Чуватова М.В.

A Программа плавной сортировки

В данном разделе приведена, реализующая плавную сортировку.

Специальный символ @ используется для механизма меток (см. раздел 2.4). Если строка начинается с этого символа, значит адрес этой команды будет подставлен вместо любого упоминания этой метки. Если слово, начинающееся с этого символа, используется в качестве операнда, вместо него будет поставлен адрес команды, строка с которой начинается с этой метки.

```
MOV 0x50 HSP
LD 0x70 CR
STEP 1 0
HPUSH RA
LINK @ret_1
@nh MOV 0x50 RA
SUB HSP RA RA
BL2 @nh_c1
HPOP RA
HPOP RB
SUB RB RA RA
BL2 @nh_c2
ADD HSP 2 HSP
B @nh_c1
@nh_c2 ADD RB 1 RB
HPUSH RB
B @rs
@nh_c1 SUB HSP 1 RB
LD RB RA
BL2 @nh_c3
MOV 1 RA
HPUSH RA
B @rs
@nh_c3 MOV 0 RA
HPUSH RA
@rs ST 0x58 HSP
MOV AP ASR
@rs_st1 MOV 0x50 RA
SUB HSP RA RA
BL2 @rs_st1e
HPOP RA
BL2 @rs_c1
MOV 1 AR1
SUB RA 2 RB
L RB AR2
ADD AR2 1 AR2
LDAW 0 AR1
LDAW 1 AR2
BCMP @rs_c0
MOV RC RA
MOV RD RB
@rs_c0 MOV 0 AR2
LDAW 1 AR2
BCMP @rs_c2
MOV RC RA
MOV RD RB
B @rs_c2
@rs_c1 MOV 0 AR1
```

```
LDAW 0 AR1
@rs_c2 LD HSP RC
L RC AR2
LDAW 1 AR2
BCMP @rs_st2
MOV 0 AR1
LDAW 0 AR1
STAW 0 AR2
STAW 1 AR1
SUB ASR AR2 ASR
B @rs_st1
@rs_st1e STEP 0 1
@rs_st2 LD HSP RA
MOV 0x59 HSP
ST HSP RA
@rs_st2l BL2 @rs_e
MOV 1 AR1
SUB RA 2 RB
L RB AR2
ADD AR2 1 AR2
LDAW 0 AR1
LDAW 1 AR2
BCMP @rs_c3
MOV RC RA
MOV RD RB
MOV AR2 AR1
LD HSP RD
SUB RD 1 RD
B @rs_c4
@rs_c3 LD HSP RD
SUB RD 2 RD
@rs_c4 SUB ASR AR1 RC
HPUSH RD
HPUSH RC
MOV 0 AR2
LDAW 1 AR2
BCMP @rs_c5
B @rs_e
@rs_c5 STAW 1 AR1
STAW 0 AR2
HPOP ASR
HPOP RA
B @rs_st2l
@rs_e LD 0x58 HSP
RET
@ret_1
STEP 1 0
BEOA @dh
B @nh
@pdh STEP 1 0
@dh MOV AP RA
BL2 @end
STEP 1 1
HPOP RA
BL2 @dh
SUB RA 1 RA
```

```
HPUSH RA
SUB RA 1 RA
HPUSH RA
STEP 0 1
LINK @ret_2
L RA RA
ADD RA 1 RA
SUB AP RA AP
B @rs
@ret_2 LD HSP RA
L RA RA
ADD AP RA AP
STEP 0 0
LINK @pdh
B @rs
@end OUT 0
MOV 0 AR1
MOV 0 AP
@out_1 MOV AP ASR
LDAW 0 AR1
OUT 1
STEP 1 0
BEOA @end
B @out_1
```

B Образ памяти для lookup table

```
v2.0 raw
2 1 0 0 104 10 11 0
48 91 0 0 1a0 11
```