# Methods

## Ячейки

Каждая ячейка, кроме 8ми бит памяти хранит флаг, является ли она активной. В каждый момент времени есть только одна активная ячейка, которая исполняет приходящие по шине команды. Это позволяет легко реализовать Brainfuck команды чтения, записи, инкремента и декремента.

Проблема возникает с командами переключения активной ячейки. Надо чтобы не только активная ячейка перестала быть активной, но и соседняя ячейка с правильной стороны активировалась. При этом первая и последняя ячейки не должны позволять активировать ячейку слева и справа, соответственно.

Решение этой проблемы заключается в последовательном размещении микросхем ячеек памяти. Для того чтобы переключать текущую активную ячейку, все микросхемы ячеек подключаются последовательно друг к другу, западным краем к восточному, крайним восточным краем к процессору. Активная ячейка посылает сигнал влево и вправо своим соседям о том, что она активная. Также каждая ячейка обнаруживает неподключенные контакты, по которым она понимает что она является первой или последней ячейкой в цепочке.

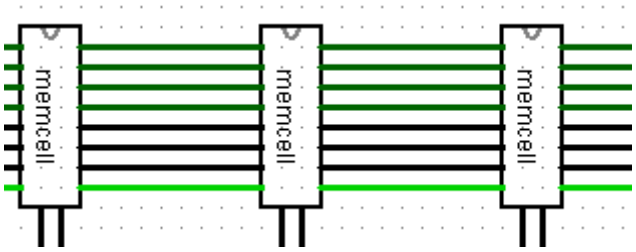


*Рисунок 1: Схема подключения ячеек*

Такая схема подключения позволяет ячейкам видеть окружающую конфигурацию и операция переключения активной ячейки выполняется одновременно на двух ячейках: той которая перестает быть активной и той, которая становится активной.

Для упрощения схемы здесь и далее используются средство Logisim «тоннели». Они позволяют автоматически соединить разные места схемы, к которым присоединена метка с общим названием.

## Шины

У нас имеются 4 сквозных шины: это DATA IN, DATA OUT, CMD и CLK.

DATA IN – входные данные, поступающие при команде записать число в ячейку. Для того, чтобы не было неопределенного состояния, подается 0 через согласующий резистор.

DATA OUT – шина выхода ячейки, при команде чтения активная ячейка выводит свое значение в эту шину.

CMD – шина для текущей команды, подаваемой процессором. Если ячейка активна, то она ее выполняет. Для того, чтобы не было неопределенного состояния, подается 0 через согласующий резистор.

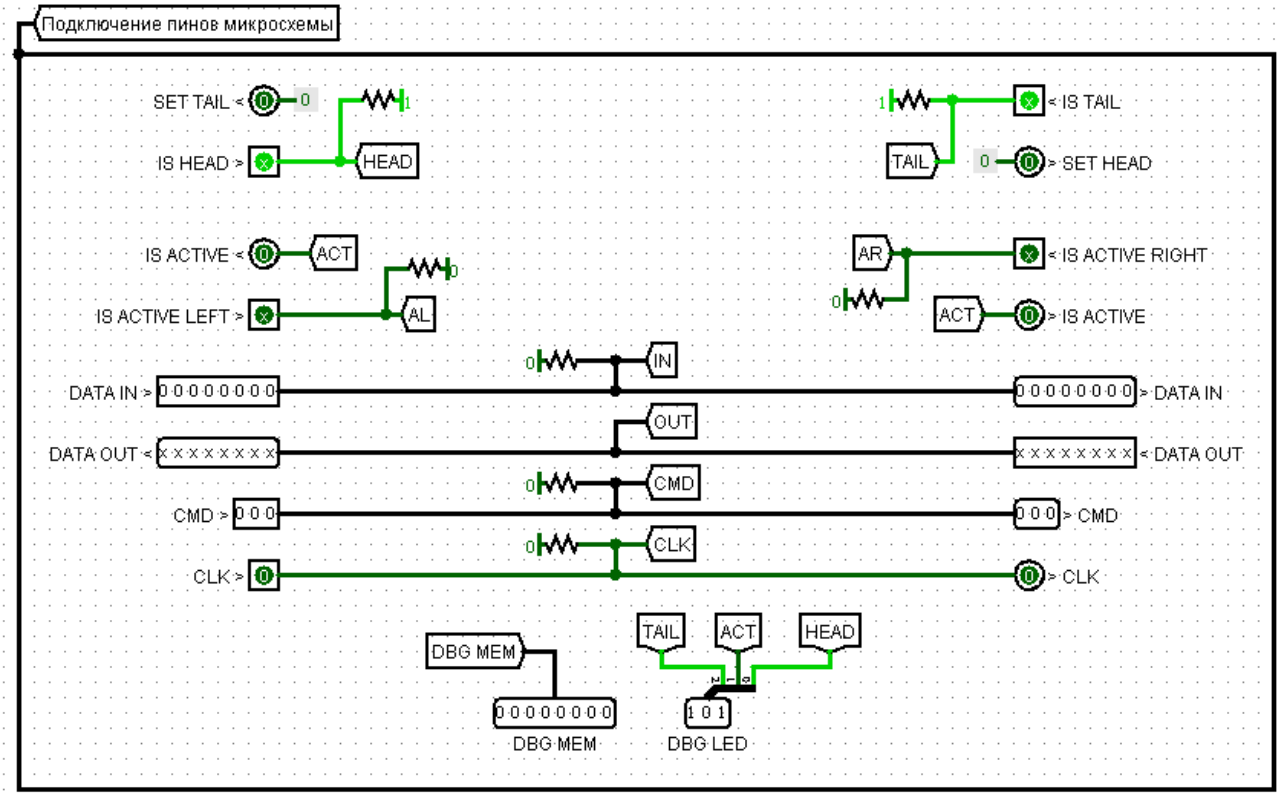


*Рисунок 2: Подключение пинов микросхемы*

CLK – шина тактового сигнала. Для того, чтобы не было неопределенного состояния подается 0 через согласующий резистор.

## Голова и хвост

Контакты SET TAIL/IS TAIL и SET HEAD/IS HEAD.

Голова (первая ячейка) и хвост (последняя) подключены к 1 через согласующий резистор. То есть мы считаем, что мы голова и хвост, пока не получим «глушащий» сигнал от соседней ячейки.

Все ячейки памяти подают сигнал налево нулевой сигнал, чтобы когда соседняя ячейка получила сигнал справа она поняла, что не является хвостовой. Причем если она не получает этого сигнал, то она является хвостовой. Все ячейки подают направо нулевой сигнал, чтобы ячейки могли определить не являются ли они головой.

## Смена активности

Контакты IS ACTIVE/IS ACTIVE LEFT/IS ACTIVE RIGHT.

Также мы подаем сигнал об текущей активности ячейки и проверяем является ли активной слева ячейка или справа. Это нужно для того, чтобы при смене активности в направлении соседней ячейки она делала себя активной.

## Отладка

Также, чтобы проверить правильность работы ячеек и посмотреть ее текущее состояние, добавлены отладочные выходы из ячейки: значение счетчика в ячейке и проверка флагов: является ли эта ячейка головой, активной и хвостом.

Для проверки текущего значения ячейки использовалось два шестнадцатеричных индикатора. Для проверки флагов использовалось светодиодная матрица 3x1 (слева направо: голова, активность, хвост).

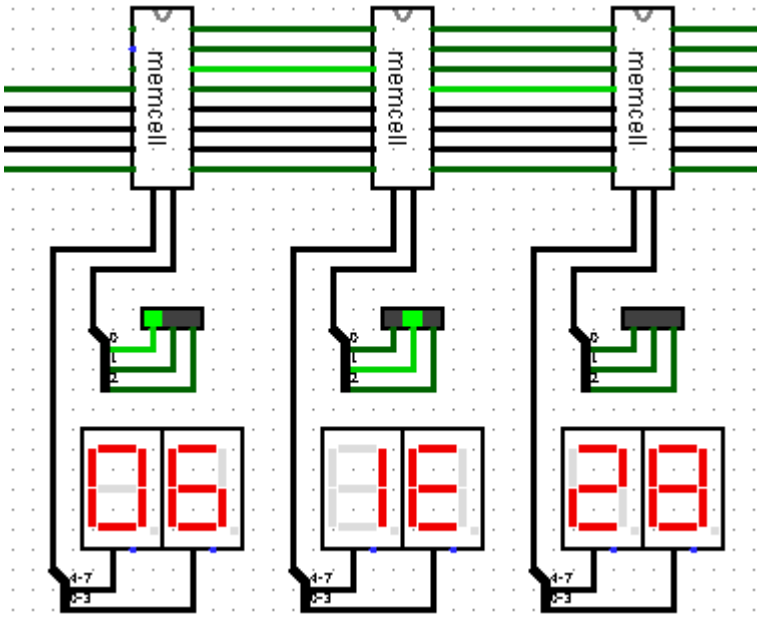


*Рисунок 3: Отладочные выходы*

## Автомат для ячейки

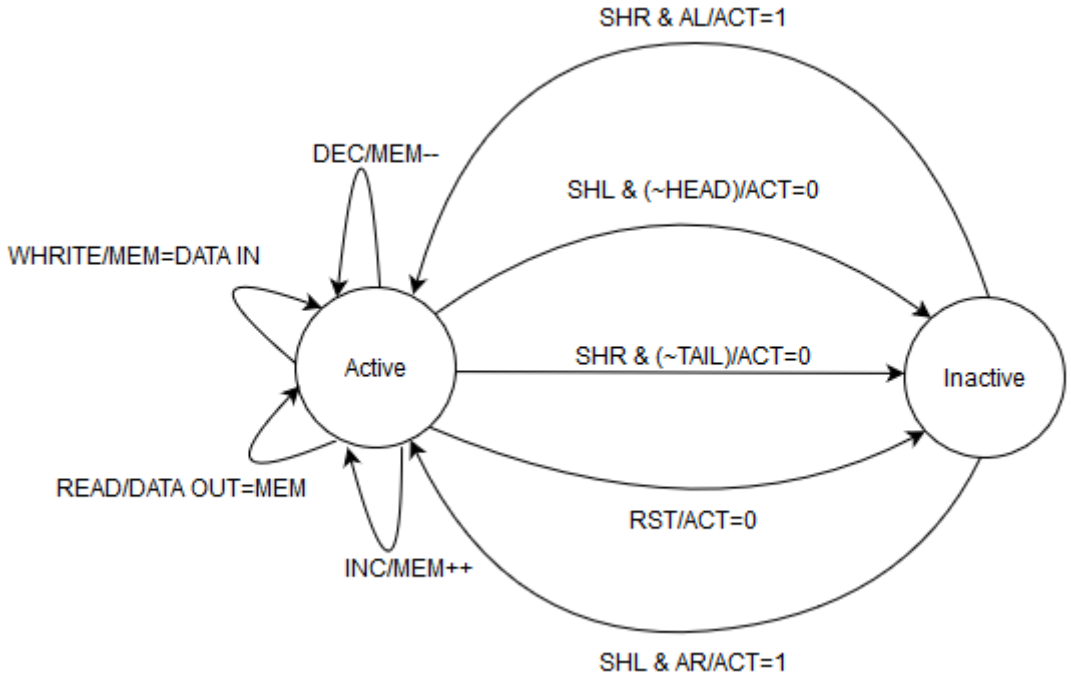


*Рисунок 4: Автомат для ячейки*

Автомат содержит всего два состояния: активное – ячейка принимает команду, и неактивное – ячейка не принимает команду.

Из-за того, что при смене активной ячейки сигнал активности одновременно и меняется у активной ячейки, и является входным для соседних ячеек, могут возникнуть конфликты. Для того чтобы их не было, мы используем MS-триггер, который фиксирует состояние на подъеме сигнала, а меняет на падении.

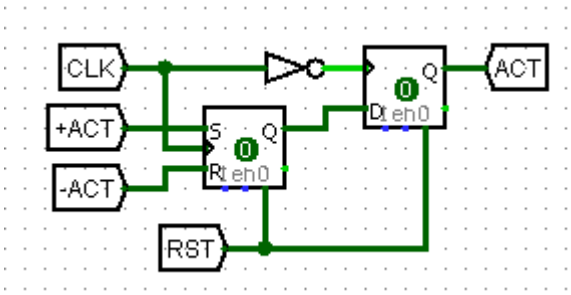


*Рисунок 5: MS-триггер активности*

Счетчик Logisim, который используется для хранения значения ячейки, поддерживает инкремент, декремент и непосредственную запись значения. Сигналы COUNT и LOAD управляют его поведением, а сигнал WRITE открывает ключ для записи значения. Эти сигналы являются выходными для автомата.

OUT – выходной сигнал. Значение 1 означает открытие ключа, который соединяет выходное значение в счетчике с шиной DATA OUT.

ACT – выходной сигнал, новое значение активности. Также его значение совпадает с номером состояния.

Ячейка поддерживает семь команд (шесть команд Brainfuck и команду RST для сброса в 0), поэтому для CMD используется 3х битное кодирование.

- 0 0 1 – RST, сброс ячейки в 0, неактивное состояние
- 0 1 0 – SHL, активация ячейки слева
- 0 1 1 – SHR, активация ячейки справа
- 1 0 0 – READ, чтение данных из активной ячейки в шину
- 1 0 1 – WRITE, запись данных из шины в активную ячейку
- 1 1 0 – INC, инкремент значения текущей ячейки
- 1 1 1 – DEC, декремент значения текущей ячейки

Обозначим хранимое состояние активности как Q, 1 – активна, 0 – неактивна.

| CMD | AR | AL | H | T | Q | COUNT | LOAD | WRITE | OUT | ACT |
|---|---|---|---|---|---|---|---|---|---|---|
| INC | ~ | ~ | ~ | ~ | 1 | 1 | 0 | 0 | 0 | 1 |
| DEC | ~ | ~ | ~ | ~ | 1 | 1 | 1 | 0 | 0 | 1 |
| READ | ~ | ~ | ~ | ~ | 1 | 0 | 0 | 0 | 1 | 1 |
| WRITE | ~ | ~ | ~ | ~ | 1 | 0 | 1 | 1 | 0 | 1 |
| SHR | ~ | ~ | ~ | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| SHR | ~ | 1 | ~ | ~ | 0 | 0 | 0 | 0 | 0 | 1 |
| SHL | ~ | ~ | 0 | ~ | 1 | 0 | 0 | 0 | 0 | 0 |
| SHL | 1 | ~ | ~ | ~ | 0 | 0 | 0 | 0 | 0 | 1 |
| RST | ~ | ~ | ~ | ~ | 1 | 0 | 0 | 0 | 0 | 0 |

Можно заметить, что удобнее разложить CMD1 и CMD2 (старший и средний биты CMD) при помощи двоичного декодера в 4 переменные SHIFT, RW, ARITH и ZERO.

| CMD0 | SHIFT | RW | ARITH | ZERO | AR | AL | H | T | Q | COUNT | LOAD | WRITE | OUT | ACT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | ~ | ~ | ~ | ~ | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | ~ | ~ | ~ | ~ | 1 | 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | ~ | ~ | ~ | ~ | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | ~ | ~ | ~ | ~ | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | ~ | ~ | ~ | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | ~ | 1 | ~ | ~ | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | ~ | ~ | 0 | ~ | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | ~ | ~ | ~ | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | ~ | ~ | ~ | ~ | ~ | 0 | 0 | 0 | 0 | 0 |

Тогда можно получить следующие уравнения для выходов:
$\mathrm{WRITE} = \mathrm{CMD0} \wedge \mathrm{RW} \wedge \mathrm{Q0}$
$\mathrm{OUT} = \neg\,\mathrm{CMD0} \wedge \mathrm{RW} \wedge \mathrm{Q0}$
$\mathrm{LOAD} = (\mathrm{CMD0} \wedge \mathrm{ARITH} \wedge \mathrm{Q0}) \vee (\mathrm{CMD0} \wedge \mathrm{RW} \wedge \mathrm{Q0})$
$\mathrm{COUNT} = \mathrm{ARITH} \wedge \mathrm{Q0}$

Построим таблицу переходов:

| CMD0 | SHIFT | RW | ARITH | ZERO | AR | AL | H | T | Q | ACT |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | ~ | ~ | ~ | ~ | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | ~ | ~ | ~ | ~ | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | ~ | ~ | ~ | ~ | ~ | 0 |
| ~ | 0 | ~ | ~ | 0 | ~ | ~ | ~ | ~ | 0 | 0 |
| ~ | 0 | ~ | ~ | 0 | ~ | ~ | ~ | ~ | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | ~ | ~ | ~ | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | ~ | ~ | ~ | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | ~ | ~ | 0 | ~ | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | ~ | ~ | 1 | ~ | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | ~ | 0 | ~ | ~ | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | ~ | 1 | ~ | ~ | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | ~ | ~ | ~ | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | ~ | ~ | ~ | 1 | 1 | 1 |

При SHIFT=1
$D = CMD\,0 \wedge AL \wedge \neg\, Q \vee \neg\, CMD\,0 \wedge AL \wedge \neg\, Q \vee \neg\, CMD\,0 \wedge \neg\, H \wedge Q \vee CMD\,0 \wedge T \wedge Q$

Также при ZERO = 1 и CMD0 = 1 ячейка памяти сбрасывает свое состояние, передавая «1» на RST-входы счетчика и MS-триггера.

В остальных случаях D не меняется.

# Процессор

В первом приближение автомат CPU можно представить в виде следующего абстрактного автомата.

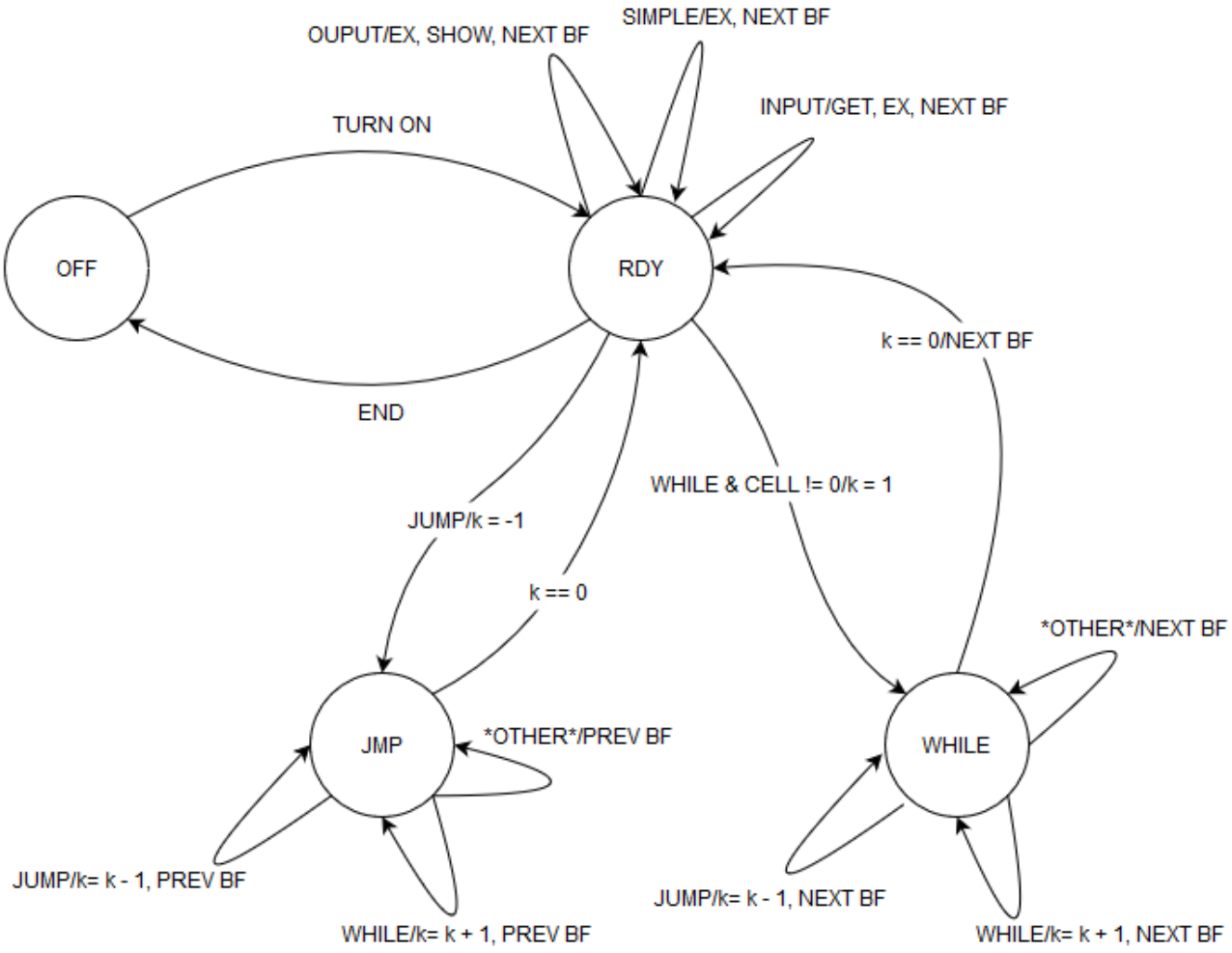


*Рисунок 6: Автомат CPU (первое приближение)*

Изначально мы находимся в выключенном состоянии OFF. По сигналу включения TURN ON переходим в состояние готовности RDY. В этом состоянии мы исполняем большинство команд: простые команды SIMPLE (INC, DEC, LEFT, RIGHT), ввод с клавиатуры INPUT , вывод на терминал OUTPUT. При встрече с командами цикла WHILE или JUMP мы переходим в специальные состояния, в которых ищем парную им команду. При достижении конца ленты команд автомат переходит в исходное состояние OFF.

Рассмотрим более детально работу автомата.

Ячейка памяти исполняет команды на подъеме тактового сигнала, при этом она использует значения на шине DATA IN и CMD. Для того, чтобы не было конфликтов при одновременном изменении значений и исполнении команды, можно использовать буферный регистр, который будет менять свое значение на падении тактового сигнала. Таким образом исполнение ячейкой команды происходит в два этапа: выставление необходимого значения в буферные регистры и отсылка тактового сигнала.

## Инициализация

Для того, чтобы начать исполнять программу на Brainfuck, во всех ячейках памяти должны быть нулевые значения, а головная ячейка памяти должна быть активной. Для этого нужно последовательно выполнить две команды: RST, которое сбрасывает все ячейки в 0 и выключает флаг активности, и SHR, для того чтобы активировать головную ячейку. При этом, для того чтобы головная ячейка исполнила команду SHR, в точке подключения к CPU ей нужно послать сигнал ACTIVE LEFT, как будто слева от нее находится активная ячейка.

Также перед запуском программы, что лента команд отмотана на первую команду. Для этого нужно послать сигнал RST TAPE в ленту.

Таким образом, стадии инициализации выглядят так:

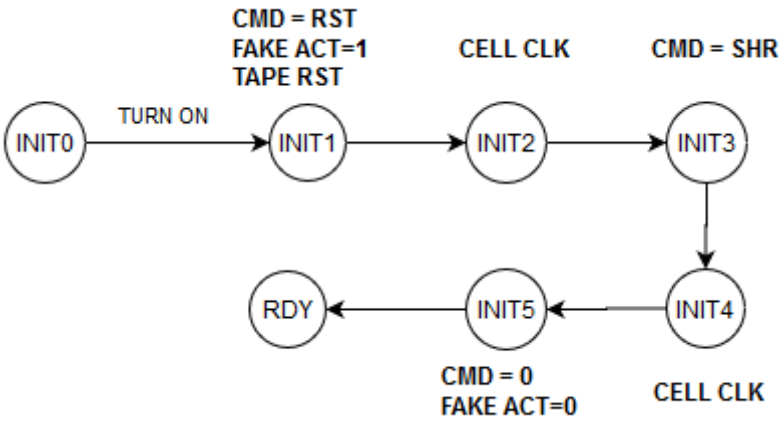


*Рисунок 7: Инициализация*

## Исполнение простой команды

Кодирование команд Brainfuck выбрано таким образом, чтобы все команды, которые ячейка может выполнить самостоятельно, совпадали с тремя младшими битами соответствующей команды Brainfuck. Поэтому исполнение команд SHR, SHL, INC, DEC устроено очень просто: в буферный регистр CMD записываются три младших бита команды Brainfuck, после чего в шину CELL CLK посылается тактовый импульс. После исполнения команды в ленту посылается сигнал перейти на следующую команду.

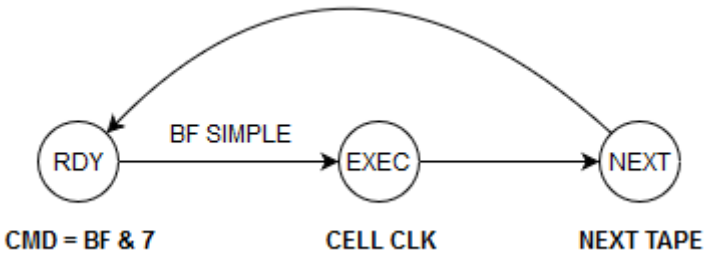


*Рисунок 8: Простые команды*

## Печать на экран

В отличие от простых команд, для печати на экран необходимо выполнить еще один шаг. После того как ячейка выполнит команду, и на шине от FROM CELL появится содержимое активной ячейки, нужно послать тактовый импульс в терминал. После этого в ленту посылается сигнал перейти на следующую команду.

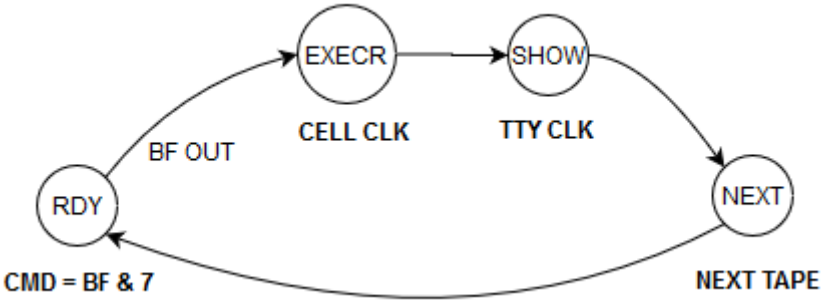


*Рисунок 9: Команда OUT*

## Чтение с клавиатуры

Для команды ввода с клавиатуры нужно предусмотреть случай, когда клавиатурный буфер пуст. После того как в клавиатурном буфере будет хотя бы один символ он записывается в буферный регистр, после чего в шину CELL CLK посылается тактовый импульс, а в клавиатурном буфере удаляется символ. После этого в ленту посылается сигнал перейти на следующую команду.

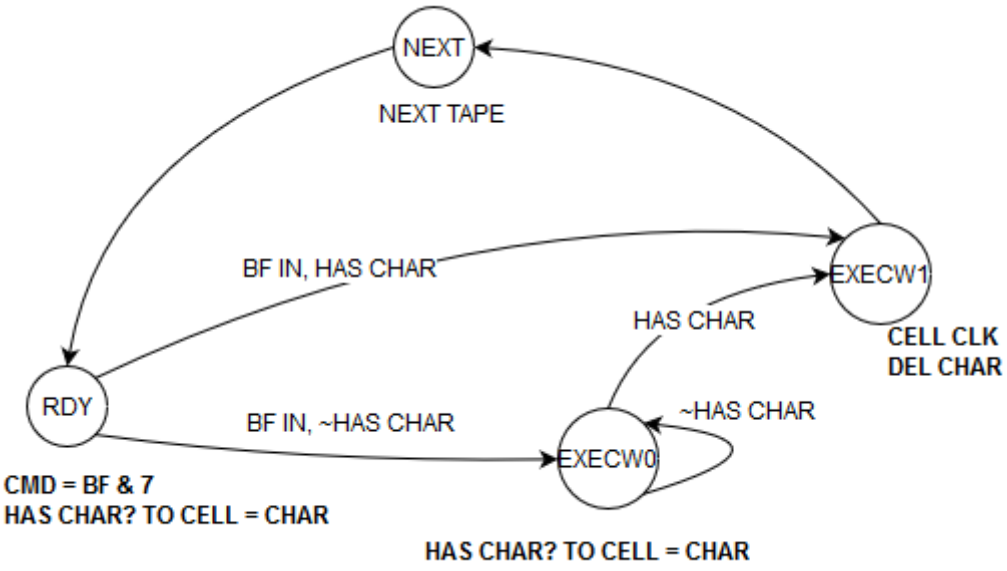


*Рисунок 10: Команда IN*

## Поддержка циклов

Инструкция JUMP должна отмотать ленту команд назад на парную ей инструкцию WHILE (с учетом правильной скобочной последовательности). Для этого можно использовать счетчик, увеличивая его значение каждый раз, когда встречается команда WHILE и уменьшая – когда встречается команда JUMP.

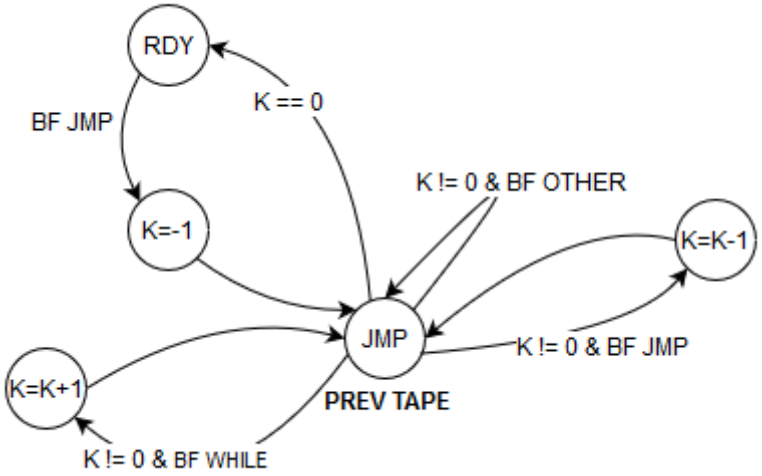


*Рисунок 11: Команда JUMP*

Для инструкции WHILE требуется сначала просмотреть содержимое текущей ячейки, и если в ней 0, то промотать ленту команд сразу после парной ей инструкции JUMP, иначе продолжать исполнение со следующей после WHILE команды.

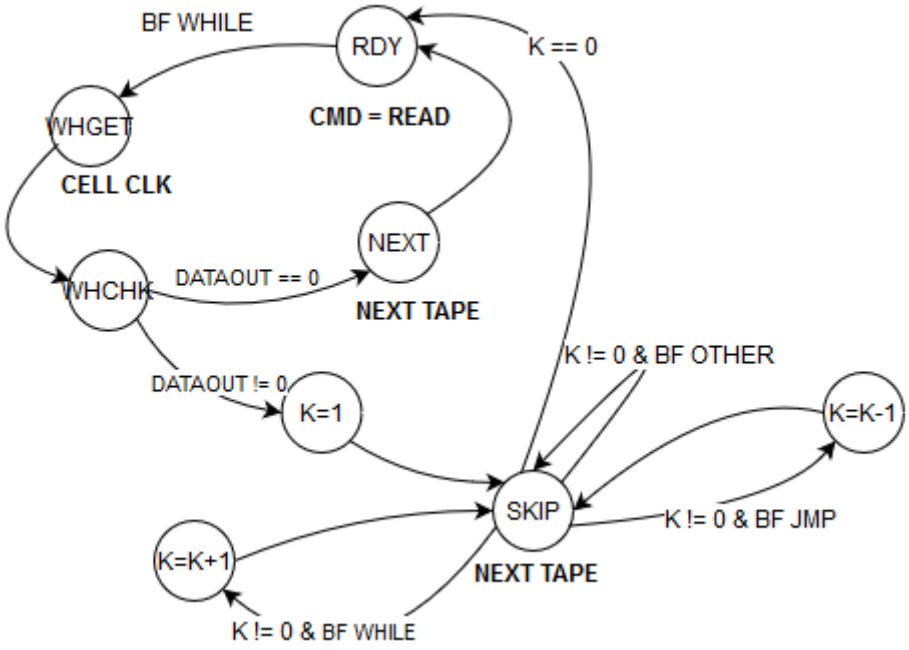


*Рисунок 12: Команда WHILE*

В блоках для WHILE и JUMP видны схожие состояния, которые занимаются подсчетом скобочной последовательности. Их можно реализовать в виде отдельной схемы со счетчиком,

которая будет заниматься только вычислением баланса скобочной последовательности. Тогда при обработке команд WHILE и JUMP можно в нужный момент включать эту схему и идти по ленте пока баланс скобочной последовательности не станет нулевым.

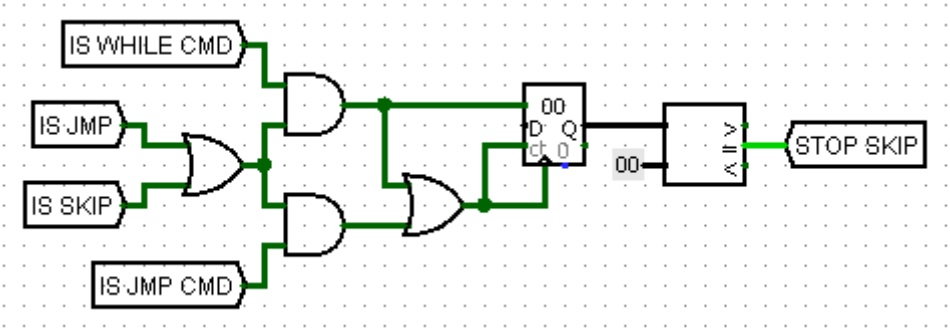


*Рисунок 13: Счетчик скобочной последовательности*

## Конец ленты

Команда END сигнализирует о том, что команды в ленте закончились, в отличие от остальных команд она единственная имеет старший бит (STOP FLAG) равным единице. При наличии этого бита автомат сразу переходит в состояние INIT0.


*Рисунок 14: Команда END*

## Итоговый автомат

Скомбинировав отдельные участки мы получаем итоговый автомат для блока процессора:

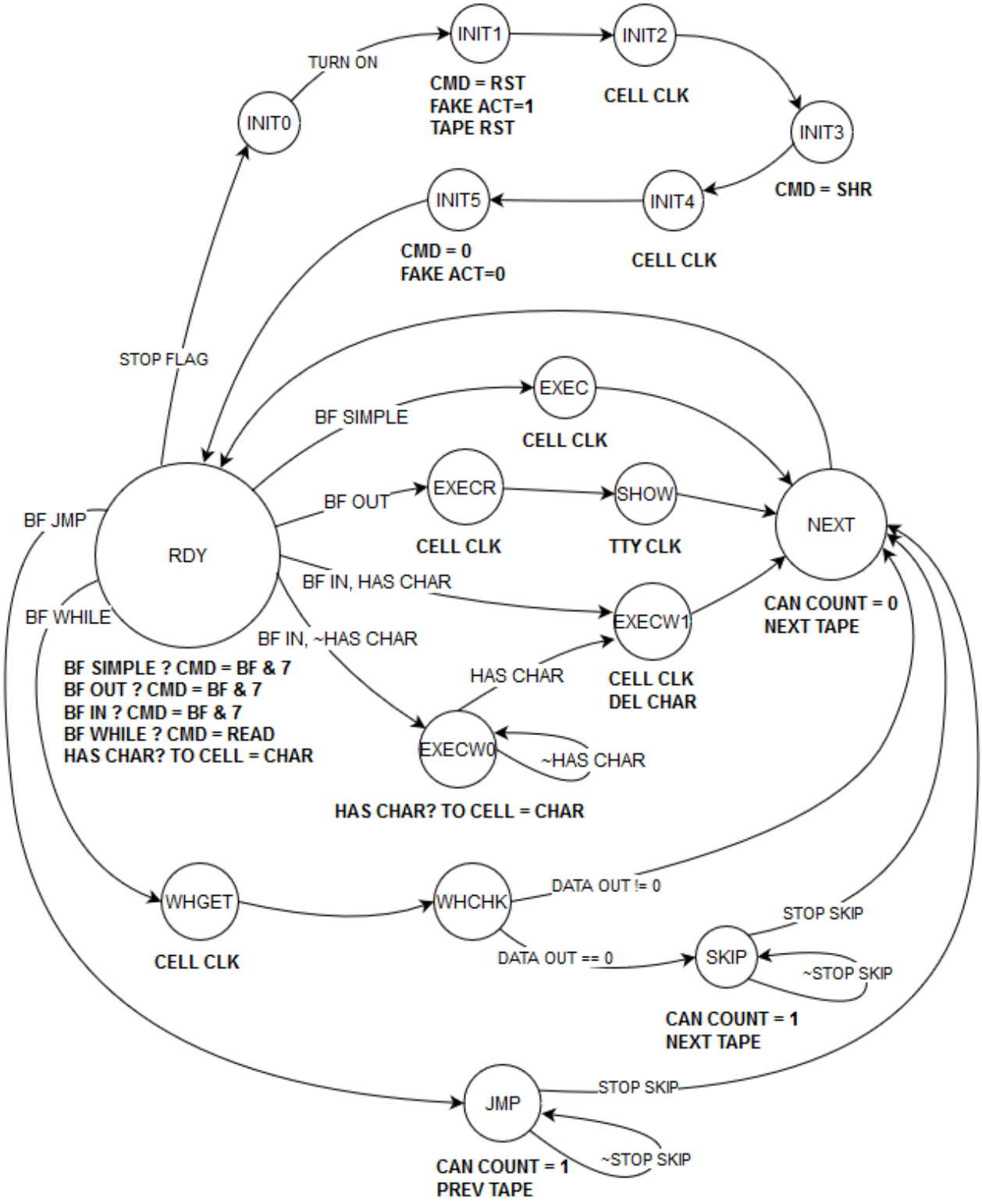


*Рисунок 15: Итоговый автомат*

У полученного автомата 17 состояний, поэтому для хранения состояний необходимо 5 D-триггеров, либо, для простоты, один 5-битовый регистр. Входными сигналами для этого автомата являются:

- TURN ON – сигнал включения
- HAS CHAR – сигнал наличия символа в буфере клавиатуры
- CHAR (7 бит) – текущий символ клавиатурного буфера
- DATA OUT (8 бит) – значение, прочитанное из текущей активной ячейки
- STOP SKIP – сигнал счетчика скобочной последовательности о том, что последовательность команд WHILE-JUMP сбалансирована.
- BF CMD (3 бита) – закодированная команда Brainfuck
- STOP FLAG – лента указывает на команду END.

Выходные сигналы автомата используются для того, чтобы открыть те или иные ключи соединяющие части схемы. Перечислим эти сигналы:

- SET ACT – включает буферный D–триггер FAKE ACT позволяющий сделать активной головную ячейку при инициализации.
- CLEAR ACT – выключает буферный D–триггер FAKE ACT.
- TAPE RST – сигнал для перемотки ленты команд на начало.
- CMD (3 бита) – команда для ячеек памяти, которая поступает на вход буферного регистра.
- CELL CLK – посылка тактового сигнала в ячейки по шине.
- TTY CLK – отправка тактового импульса в терминал.
- DEL CHAR – сигнал удалить текущий символ из клавиатурного буфера.
- NEXT TAPE – сигнал ленте перейти на следующую команду.
- PREV TAPE – сигнал ленте перейти на предыдущую команду.
- TO CELL (8 бит) – данные для ячейки памяти, которые поступают на вход соответствующего буферного регистра.
- CAN COUNT – сигнал, разрешающий подсчет баланса скобочной последовательности WHILE-JUMP.

Итого, в автомате 17 состояний, 27 переходов, 22 бита входных и 20 бит выходных сигналов. Поэтому построения таблицы переходов и оптимизации «в лоб» не представляется возможным. К счастью, многие сигналы используются в небольшом количестве состояний (например, сигнал TURN ON используется только в состоянии INIT0), а зависимости выходных сигналов от состояния и входных сигналов зачастую тривиальны (например, выходной сигнал TO CELL в состоянии EXECW0 формируется только из входного сигнала CHAR). Это позволяет применить творческий подход в проектировании итоговой схемы.

Закодируем состояния таким образом, чтобы отделить состояния инициализации от рабочих состояний:

При помощи декодеров по регистру состояния можно получить отдельный однобитовый сигнал о том, что автомат находится в соответствующем состоянии. Назовем такие сигналы “IS <состояние>”. Эти сигналы будет удобно использовать в различных ключах и логических элементах, связанных с поведением в нужных состояниях.

Для удобства обозначим IS SIMPLE CMD = (BF CMD == BF_INC | BF CMD == BF_DEC | BF CMD == BF_LEFT | BF CMD == BF_RIGHT), IS LIKE JMP CMD = (BF CMD == BF_WHILE | BF CMD == BF_JMP), CELL ZERO = (DATA OUT == 0).

Выпишем функции для выходных сигналов:

- SET ACT = IS INIT1
- CLEAR ACT = IS INIT5
- TAPE RST = IS INIT1
- CELL CLK = IS INIT2 $\vee$ IS INIT4 $\vee$ IS EXEC $\vee$ IS EXECR $\vee$ IS EXECW1 $\vee$ IS WHGET
- TTY CLK = IS SHOW
- DEL CHAR = IS EXECW1
- NEXT TAPE = IS NEXT $\vee$ IS SKIP
- PREV TAPE = IS JMP
- TO CELL = CHAR если HAS CHAR $\wedge$ (IS RDY $\vee$ IS EXECW0), то есть условие открывает ключ из TO CELL = CHAR в TO CELL = CHAR
- CAN COUNT = IS JMP $\vee$ IS SKIP
- $$\text{CMD} = \begin{cases} \text{BF CMD}, \text{если IS RDY} \wedge \neg \text{IS LIKE JMP CMD} \\ \text{CMD\_RST}, \text{если IS INIT1} \\ \text{CMD\_SHR}, \text{если IS INIT3} \\ 0, \text{если IS INIT5} \\ \text{CMD\_READ}, \text{если IS RDY} \wedge (\text{BF CMD} == \text{BF\_WHILE}) \end{cases}$$

Для удобства переключения между состояниями заведем средствами Logisim константы с соответствующими значениями. Назовем такие константы “ST_<состояние>”. Тогда схема перехода в новое состояние будет выглядеть как константа подключенная через ключ к регистру состояний, а условие на ключе – проверка на текущее состояние плюс, возможно, дополнительная логика.

Выпишем схемы перехода:

| Условие ключа | Константа нового состояния |
|---|---|
| IS INIT0 & TURN ON | ST_INIT1 |
| IS INIT1 | ST_INIT2 |
| IS INIT2 | ST_INIT3 |
| IS INIT3 | ST_INIT4 |

| | |
|---|---|
| IS INIT4 | ST_INIT5 |
| IS INIT5 | ST_RDY |
| IS RDY & IS SIMPLE CMD | ST_EXEC |
| IS RDY & (BF CMD == BF_OUT) | ST_EXECR |
| IS RDY & (BF_CMD == BF_IN) & HAS CHAR | ST_EXECW1 |
| IS RDY & (BF_CMD == BF_IN) & ~HAS CHAR | ST_EXECW0 |
| IS RDY & (BF_CMD == BF_WHILE) | ST_WHGET |
| IS RDY & (BF_CMD == BF_JMP) | ST_JMP |
| IS EXEC | ST_NEXT |
| IS EXECR | ST_SHOW |
| IS SHOW | ST_NEXT |
| IS EXECW0 & HAS CHAR | ST_EXECW1 |
| IS EXECW1 | ST_NEXT |
| IS WHGET | ST_WHCHK |
| IS WHCHK & CELL ZERO | ST_SKIP |
| IS WHCHK & ~CELL ZERO | ST_NEXT |
| IS SKIP & STOP SKIP | ST_NEXT |
| IS JMP & STOP SKIP | ST_NEXT |

# Лента команд

Лента команд реализована через микросхему-контроллер, к которой подключается блок ПЗУ, содержащий команды. Сигналы NEXT TAPE и PREV TAPE непосредственно увеличивают и уменьшают счетчик содержащий адрес команды, сигнал TAPE RST сбрасывает счетчик в 0, а выход данных от ПЗУ напрямую подключен к выходу BF CMD.