**Оглавление**

1. Поставленная задача3

2. Среда разработки, тестирования и отладки3

3. Алгоритм работы программы4

3.1. Описание алгоритма4

3.2. Блок-схемы 7

4. Общие сведения о программе и об аппаратной части13

4.1. Электрическая принципиальная схема 13

4.2. Используемые выводы микроконтроллера14

4.3. Используемые регистры14

5. Методика и результаты тестирования15

Приложение 1. Код программы (файл .hex)21

Приложение 2. Код программы (файл .prj)27

**1. Поставленная задача.**

В данной курсовой работе было необходимо реализовать парсер, который должен уметь распознавать среди поступающих через UART данных определенные команды, и по их распознаванию подавать определенный сигнал. Развернутая постановка задачи:

1. Необходимо сформировать 3 различные команды, которые будет распознавать парсер. (В нашем случае эти 3 команды – GO, HELP и STOP)

2. Каждая из команд должна начинаться с символа # и заканчиваться символом $.

3. Поступающие на вход через UART данные должны записываться в кольцевой буфер размером 16 байт.

4. Парсер должен брать сохраненные данные из кольцевого буфера и обрабатывать их, проводя соответствующие операции распознавания.

5. По распознаванию команды программа должна подать определенный сигнал, который дал бы понять, что команда – распознана. В нашем случае -это зажигание одного из 3х диодов, а также вывод распознанной команды через UART.

**2. Средства разработки, тестирования и отладки**

Для выполнения курсовой работы была использована программа VMLAB, тестирование и отладка программы были проведены в ней же с помощью таких инструментов, как

1. Control Panel (просмотр работы светодиодов и вывода команд через UART)

2. Registers/Flags (проверка корректной работы указателей, записываемых данных в регистры и т.д.)

3. Data Memory (проверка корректной записи данных в ОЗУ, проверка работы кольцевого буфера и т.д.)

**3. Алгоритм работы программы.**

**3.1. Описание алгоритма.**

Программа реализована следующим образом. По прерыванию, которое вызывается в случае приема данных через UART, вызывается подпрограмма по сохранению принятых с UART данных в ОЗУ микроконтроллера. Затем, во время этого же прерывания происходит обработка сохраненного символа – следующие подпрограммы определяют – является ли сохраненный символ частью какой-либо из команд или же нет, и затем принимают решение – выводить ли его через UART и зажигать ли соответствующий диод (и если да, то какой), или же нет. В вечном цикле forever происходит проверка кольцевого буфера, в который записываются все вводимые данные, на переполнение.

В этом абзаце объясняется, как по вводу 1го символа определить, является ли этот символ частью какой-либо из команд или же нет. Из постановки задачи – любая команда должна начинаться и заканчиваться спецсимволом (в нашем случае – это # и $ соответственно). То есть, начинать сравнение введенного символа с символами изначально известных 3х команд можно только после того, как был введен знак #. Ввод же знака $ будет означать, что передача команды была завершена, следовательно, можно прекращать сравнивать последующие вводимые данные с исходными командами, пока не будет введен следующий символ #.

После подпрограммы **save**, которая сохраняет поступившие данные в ОЗУ, вызываются следующие 2 подпрограммы – **search** и **search\_0**, первая из которых проверяет – является ли последний сохраненный символ знаком #, а 2я – является ли он знаком $. Помимо этого, подпрограммы также учитывают те моменты, когда символ # уже был введен (делается это с помощью флага в r18, который поднимается по вводу #, и опускается по вводу $), то есть, идет ввод непосредственно команды – в таком случае, начинают работать следующие подпрограммы по сравнению символа с командами. Если же был введен знак $ - то программа выключает все диоды (раз команда была введена – значит, какой-то из диодов точно горел), опускает флаг в r18, который поднимался по #, а также устанавливает указатель Х вновь на 1й массив (так как по прохождению 1й, 2й и 3й подпрограмм по нахождению массива указатель Х сдвигается на 1й, 2й и 3й массив, и остается там до самого конца распознавания команды. По $ - все приходит в первоначальное состояние).

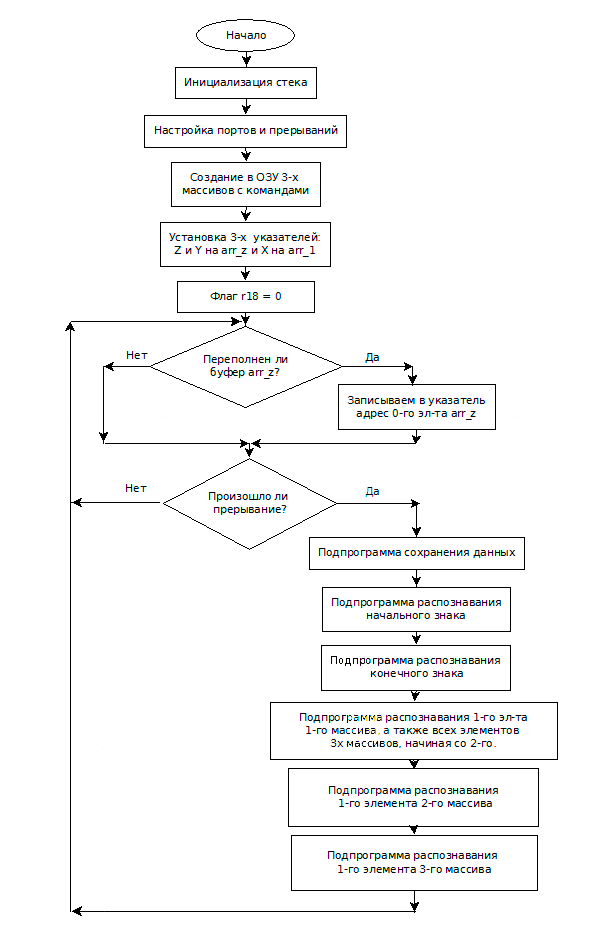
В случае если был введен знак # и последний сохраненный символ не является $ - запускается первая подпрограмма поиска **search\_1**. При самом первом прохождении данной подпрограммы она будет сравнивать вводимый символ с 1м символом 1го массива (т.к. указатель изначально стоит на 1м массиве), и, в случае его совпадения, проведет индикацию – зажжет 1й диод, а также выведет данный символ, после чего инкрементирует указатели X и Y и выйдет из прерывания. (Стоит отметить, что указателей в программе всего 3 – Z, Y и X. Первый сдвигается по мере сохранения введенных данных в arr\_z (кольцевой буфер), второй – сдвигается по мере обработки сохраненных данных в arr\_z, третий же – переходит между 1, 2 и 3 массивами (arr\_1, arr\_2, arr\_3), в каждом из которых находится соответствующая команда)

Если же последний сохраненный символ не является 1м элементом 1го массива, то указатель X переходит на начало 2го массива, и происходит переход к следующей подпрограмме – **search\_2**. Как можно было догадаться, данная подпрограмма сравнивает сохраненный символ с 1м символом 2го массива. Само сравнение и последующие действия происходят аналогично предыдущей подпрограмме, за исключением одного момента, о котором позже. Если сохраненный символ не совпадает с 1м элементом 2го массива – ставим указатель Х на 3 массив и переходим к 3й подпрограмме.

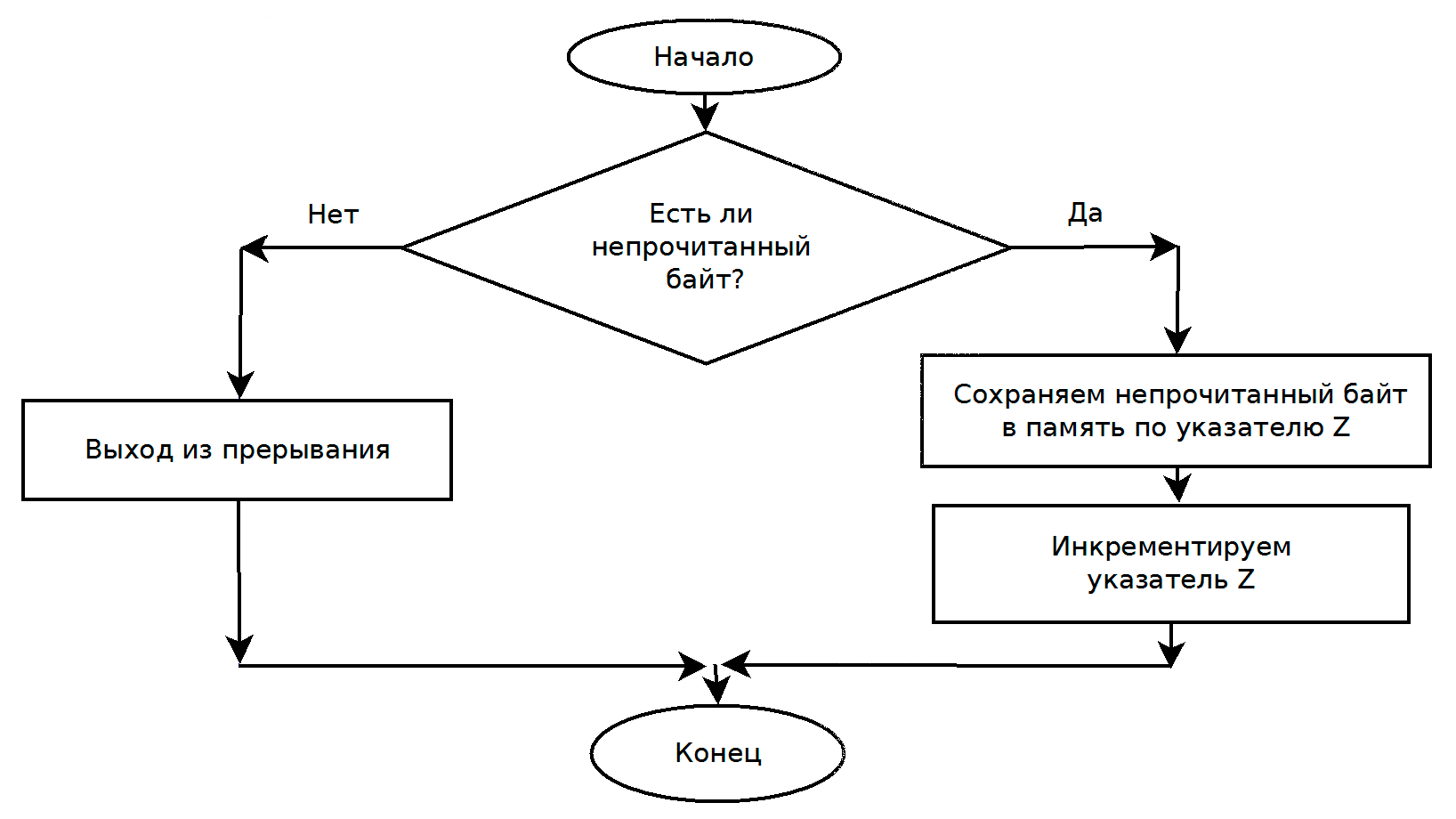
3я подпрограмма **search\_3** работает по принципу предыдущих 2х подпрограмм, однако есть одно отличие. В конце данной подпрограммы мы оставляем указатель Х на 3м массиве – ведь если данная подпрограмма работает, значит, была введена # и еще не был введен $, то есть, идет непосредственно ввод команды. Причем эта команда, по заданию, обязана быть одной из тех трех, которые были изначально записаны в 1, 2 и 3й массивы. Ситуации, при которой после # идет неизвестный набор символов – быть не может. Следовательно, раз начала работать 3я подпрограмма – это значит, что идет ввод непосредственно 3й команды (т.к. работающие до этого 2 подпрограммы не привели к выходу из прерывания).

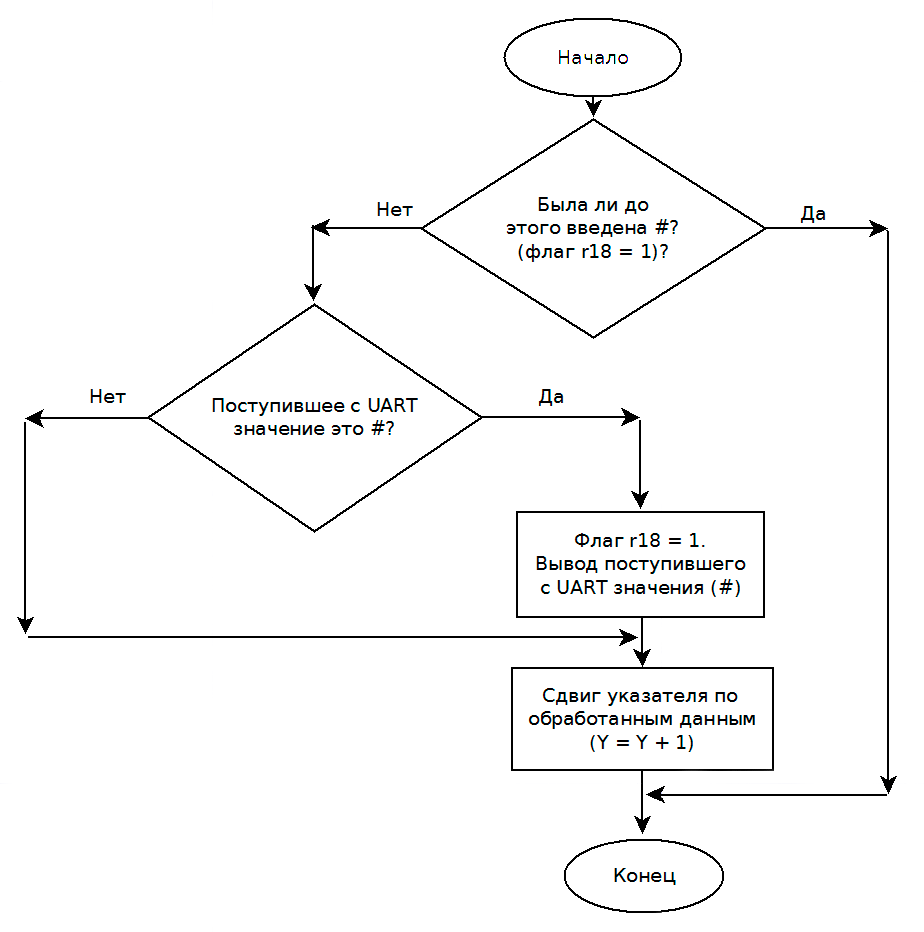
В любом случае, работа подпрограмм завершается выходом из прерывания. Стоит отметить еще один момент – 2я и 3я подпрограммы работают только при распознавании 1го элемента. Иными словами - начиная с обработки 2го элемента любого из массивов – будет работать только 1я подпрограмма, и происходит это из-за указателя Х. После распознавания 1го элемента любая из подпрограмм оставляет указатель Х на своем массиве, лишь инкрементируя этот указатель. После возврата из прерывания начнет работать вечный цикл, а затем, когда прерывание вновь будет вызвано по вводу 2го и последующих символов вводимой команды – 1я подпрограмма **search\_1** будет работать по указателю X, который в последний раз как раз и был установлен на массив, где было совпадение по 1й букве. Из этого возникает некоторого рода ограничение – данная программа не будет работать, если хотя бы 2 из 3х команд будут начинаться с 1й буквы. Из-за данного нюанса в 1й подпрограмме пришлось сделать отдельную проверку на ввод 1го символа 1го массива – только в этом случае будет зажигаться 1й диод.

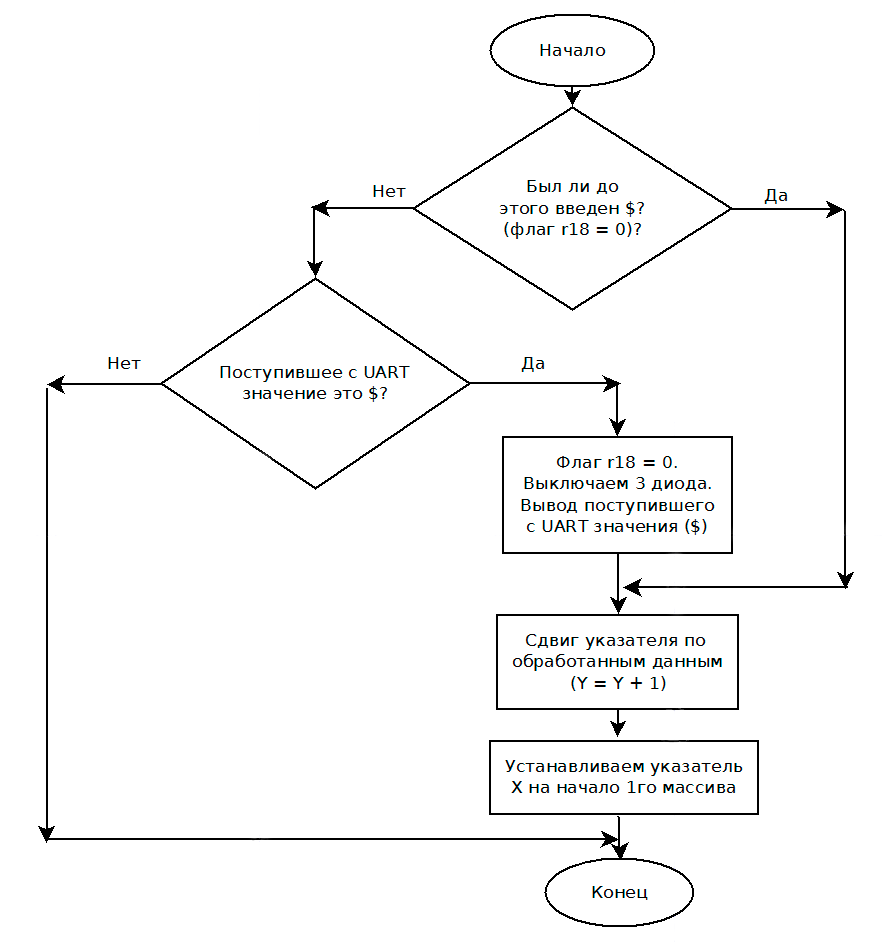
**3.2. Блок-схемы.**

****

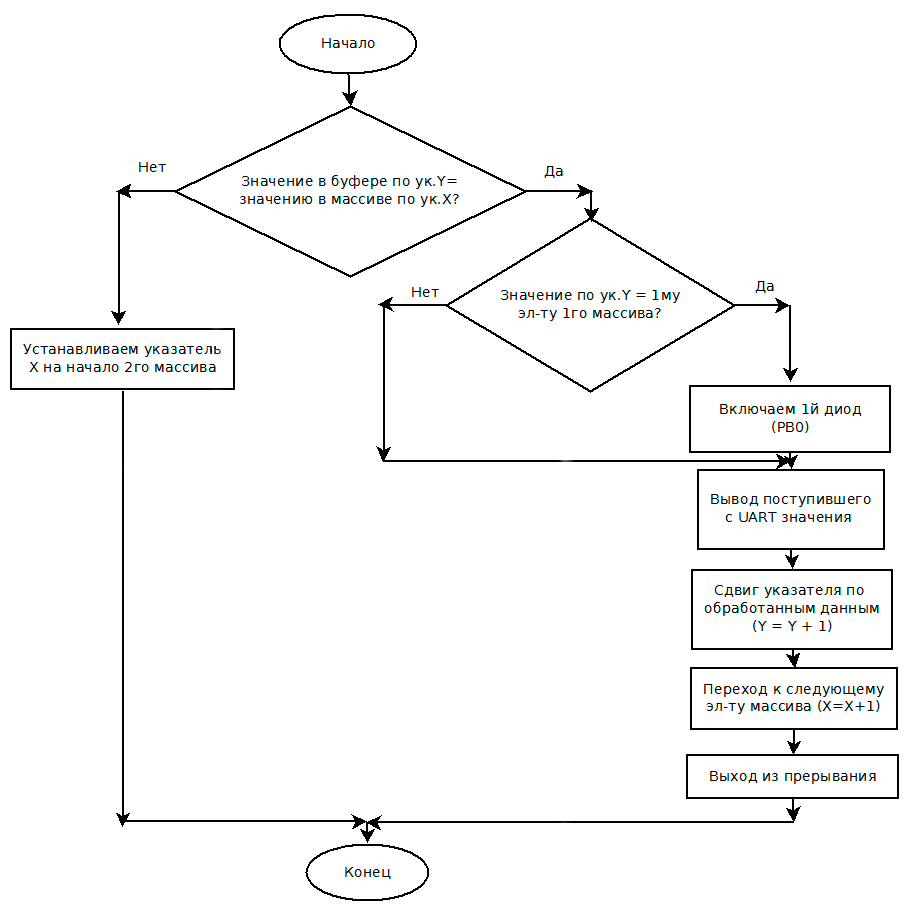
*Рис.1. Общая блок-схема алгоритма.*



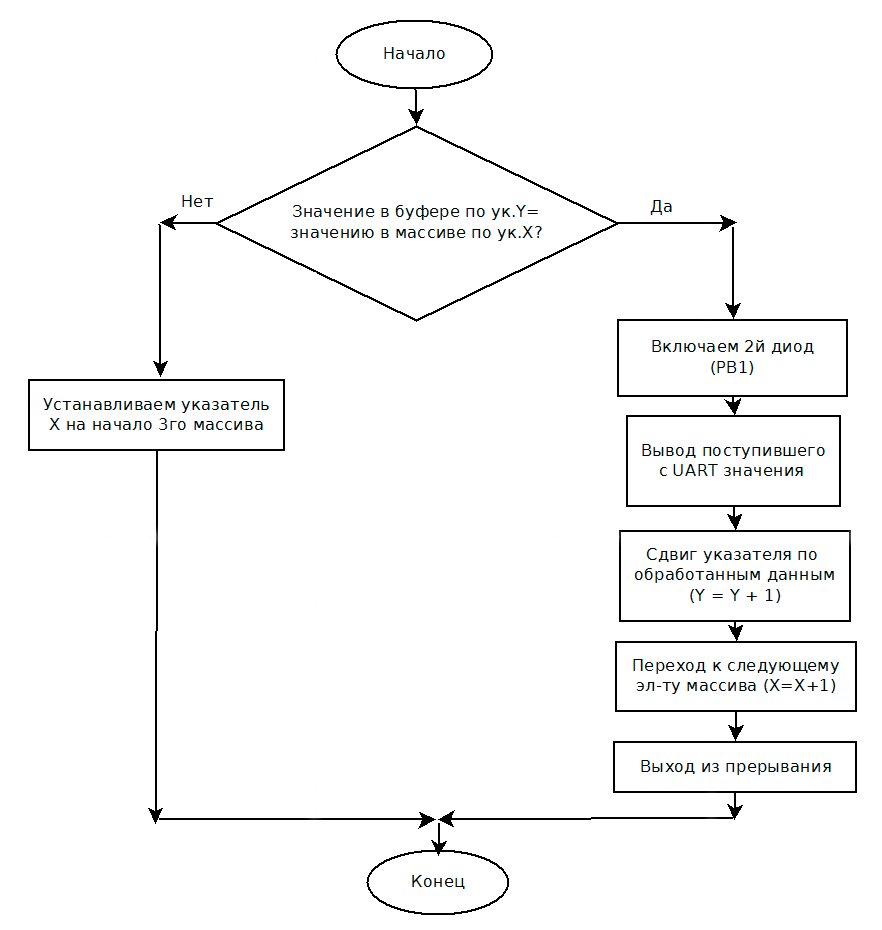
*Рис.2. Блок-схема подпрограммы сохранения данных* ***(save)****Рис.3. Блок-схема подпрограммы распознавания начального знака #* ***(search)***



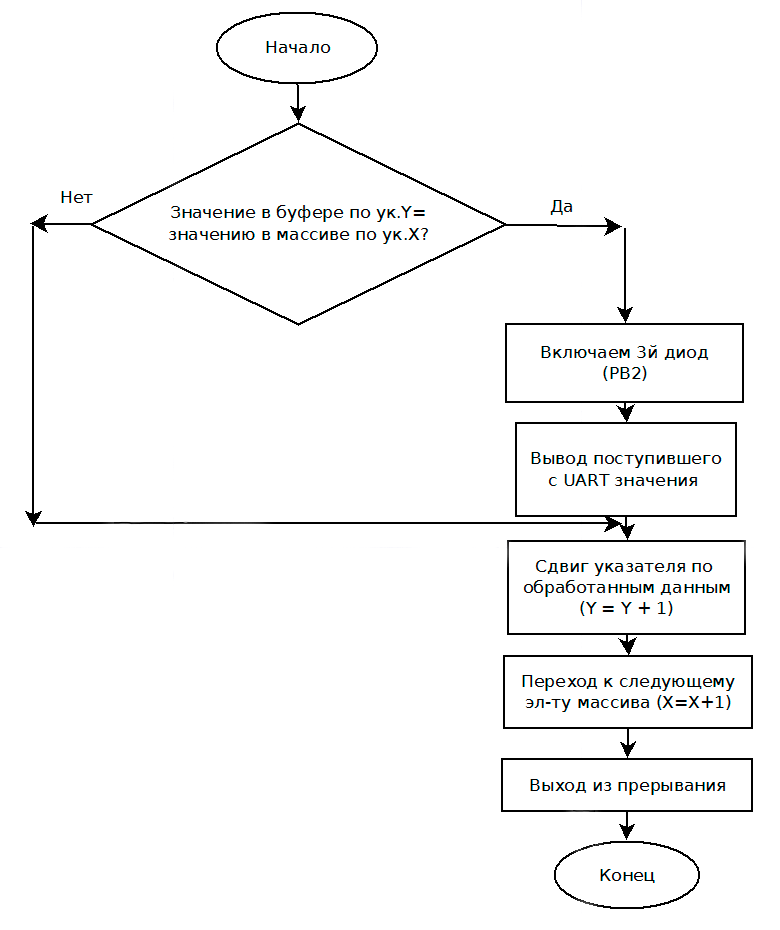
*Рис.4. Блок-схема подпрограммы распознавания конечного знака $* ***(search\_0)***



*Рис.5. Блок-схема подпрограммы распознавания 1го элемента 1го массива, а также всех элементов 3х массивов, начиная со 2го* ***(search\_1)***

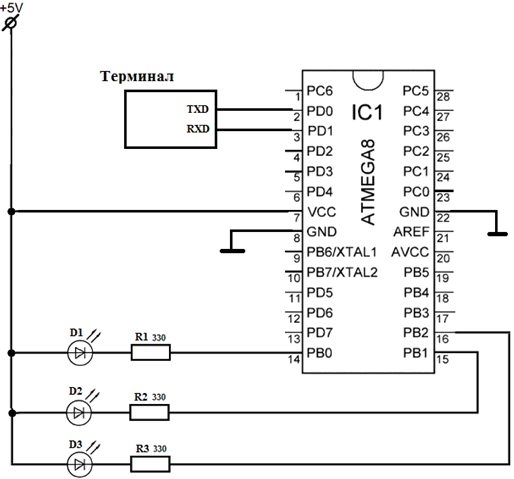


*Рис.6. Блок-схема подпрограммы распознавания 1го элемента 2го массива* ***(search\_2)***



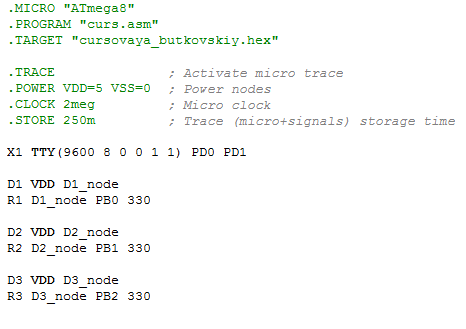
*Рис.7. Блок-схема подпрограммы распознавания 1го элемента 3го массива* ***(search\_3)***

**4. Общие сведения о программе и об аппаратной части. 4.1. Электрическая принципиальная схема.**



*Рис.8. Электрическая принципиальная схема.*

Данная электрическая схема также была реализована в программе VMLAB с помощью следующего кода в проектном файле:



*Рис.9. Код в проектном файле (.prj)*

В состав электрической схемы входят:

* Микроконтроллер AVR ATmega8
* Терминал UART
* 3 светодиода (D1, D2 и D3)
* 3 резистора (R1, R2 и R3) сопротивлением 330 Ом (Выбрано такое значение, потому что номинальный ток на диодах - 15-20мА, а чтобы получить такой ток от напряжения 5В, необходимо выбрать такое значение резистора, которое подошло бы по закону Ома - например, 330 Ом)

**4.2. Используемые выводы микроконтроллера:**

* PB0 – для соединения со светодиодом D1
* PB1 – для соединения со светодиодом D2
* PB2 – для соединения со светодиодом D3
* PD0 (RXD) – подключение терминала UART, приемник
* PD1 (TXD) – подключение терминала UART, передатчик

**4.3. Используемые регистры:**

1. DDRx – регистр направления передачи данных. Если бит установлен в 1, то соответствующий вывод порта является выходом, если же сброшен в 0 – входом. С помощью данного регистра выводы PB0, PB1 и PB2 устанавливаются в режим выхода

2. PORTx - регистр, устанавливающий напряжение на выводах Px0-Px7. Если бит установлен в 1, на выводе устанавливается напряжение высокого уровня. Если бит сброшен в 0, на выводе устанавливается напряжение низкого уровня. В данной курсовой работе регистр используется, чтобы включать и выключать светодиоды D1, D2 и D3 (1 - выкл., 0 - вкл.).

3. UBRRL – регистр, который используется для установки скорости передачи данных по UART

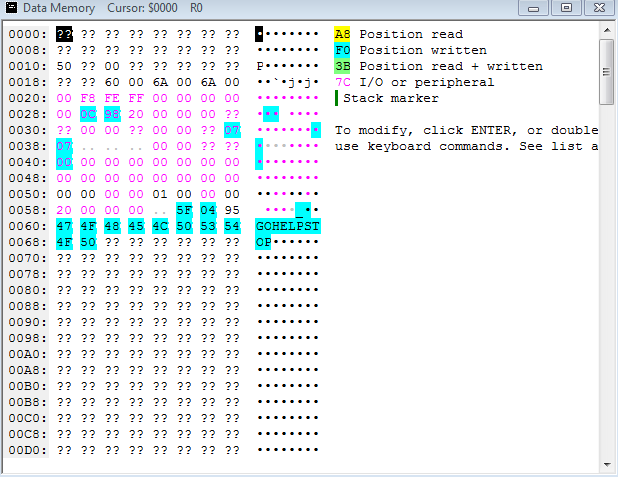
4. UCSRB – регистр управления, используется для разрешения передачи и приема данных, прерывания по приему данных.

5. UCSRC – регистр управления, используется для установки разрядности передаваемых данных.

6. SPH и SPL – регистры, которые отвечают за инициализацию стека с помощью указателей на ОЗУ.

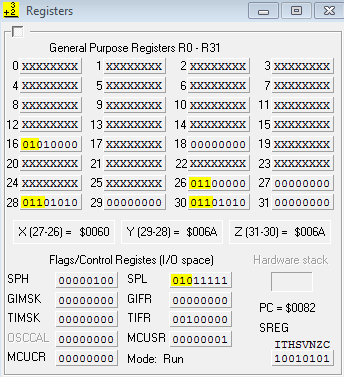
**5. Методика и результаты тестирования.**

Как уже было упомянуто в пункте “Средства разработки, тестирования и отладки” – всего для тестирования и отладки использовались 3 инструмента: Control Panel, Registers/Flags и Data Memory. Рассмотрим показания последних 2х сразу после запуска программы.



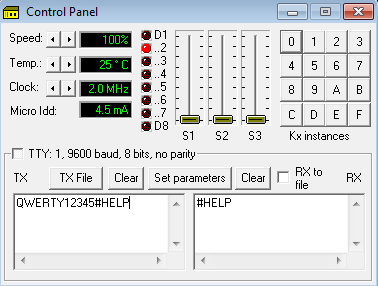
*Рис.10. Данные в ОЗУ сразу после запуска программы.*

Программа после запуска записала все 3 команды – GO, HELP и STOP в некую область оперативной памяти.



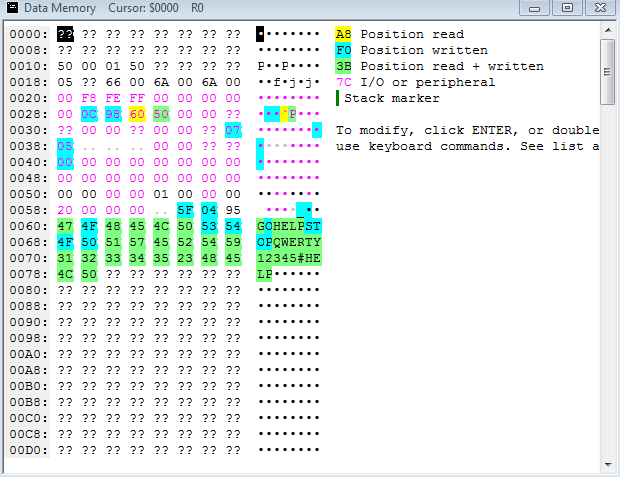
*Рис.11. Данные в регистрах сразу после запуска программы.*

Сопоставляя данные в регистрах и в ОЗУ можно увидеть, что, например, указатель X указывает на начало 1го массива – на адрес $0060 в ОЗУ - там находится символ G (часть команды GO). В это же время регистры Z и Y готовы сохранять и обрабатывать поступающие в кольцевой буфер данные, и указывают на его начало по адресу $006A. Теперь попробуем ввести набор следующих символов: “QWERTY12345#HELP”:



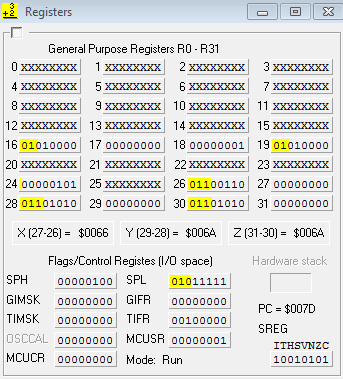
*Рис.12. Данные в контрольной панели при 1м вводе значений.*

Судя по контрольной панели – программа отрабатывает код правильно. После буквы H загорелся и будет продолжать гореть до окончания ввода команды 2й диод – как сигнал того, что вводится 2я команда (HELP). Также, в окне вывода можем увидеть, что после знака # вывелось 3 последующие введенные буквы, являющиеся частью 2й команды. Рассмотрим данные в ОЗУ:



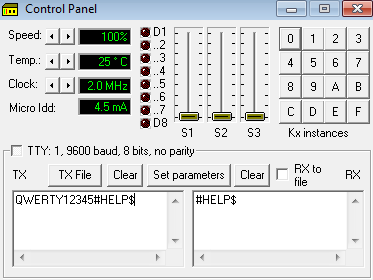
*Рис.13. Данные в ОЗУ при 1м вводе значений.*

Все наши введенные данные записались в ячейки $006A - $0079. Также стоит обратить внимание на то, что была прочитана команда HELP (символы горят зеленым цветом), находящаяся во 2м массиве. Это говорит о том, что программа действительно сравнила введенные данные с данными во 2м массиве. Также в данный момент кольцевой буфер из 16 байт полностью заполнен. Рассмотрим данные в регистрах:



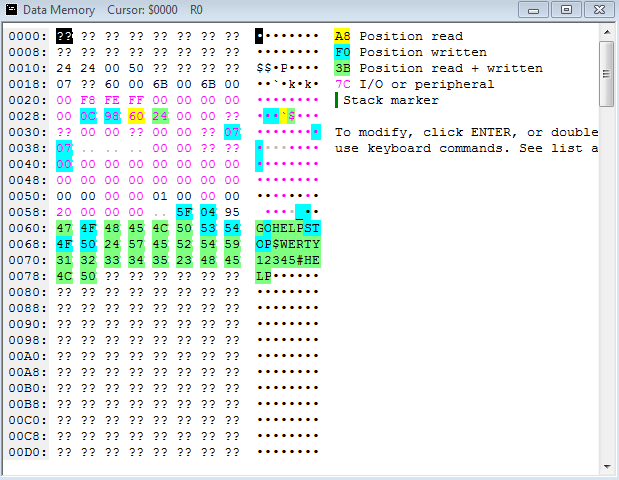
*Рис.14. Данные в регистрах при 1м вводе значений.*

Здесь стоит отметить следующие моменты. Во-первых, флаг в r18 – поднят, так как была введена # и пока что не был введен $ - команда еще в процессе ввода. Во-вторых, указатели Y и Z вновь указывают на 0й элемент кольцевого буфера – сработала проверка на переполнение в цикле forever. Теперь попробуем ввести знак $ и еще раз рассмотреть все значения.



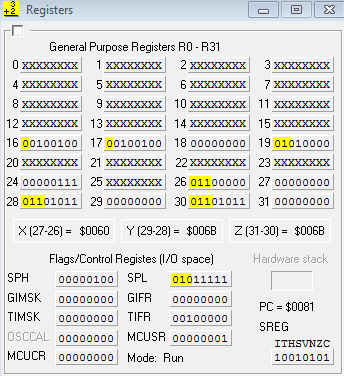
*Рис.15. Данные в контрольной панели при 2м вводе значений.*

Программа отработала штатно. 2й светодиод погас по окончанию ввода команды, знак $ - выведен.



*Рис.16. Данные в ОЗУ при 2м вводе значений.*

Здесь можно увидеть работу кольцевого буфера – самая первая введенная буква Q была перезаписана на последний введенный символ $. Значит, кольцевой буфер работает корректно.



*Рис.17. Данные в регистрах при 1м вводе значений.*

В данном окне стоит обратить внимание на опущенный флаг r18 (опустился по вводу $), а также на указатель X, который теперь вновь указывает на начало 1го массива - готов к проверке следующих символов.

Вследствие большого количества проведенных экспериментов, тестирования и отладки можно смело утверждать, что программа работает корректно и завершает работу штатно, без ошибок, а идея, которая была реализована в программе – показала свои высокие надежность и скорость выполнения.

**Приложение 1. Код программы (файл .hex).**

.include "C:\Users\75BD~1\Desktop\0B59~1\VMLAB\include\m8def.inc"

.def temp =r16

.DSEG

arr\_1: .BYTE 2 ; 1-й массив с командой GO

arr\_2: .BYTE 4 ; 2-й массив с командой HELP

arr\_3: .BYTE 4 ; 3-й массив с командой STOP

arr\_z: .BYTE 16 ; кольцевой буфер

.CSEG

reset:

rjmp start

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

rjmp save

save: ; подпрограмма сохранения поступающих данных

sbis UCSRA, RXC

reti

in r16, UDR ; сохраняем содержимое r16 в память по указателю Z,

st Z+, r16 ; после его инкрементируем

search: ; подпрограмма распознавания начального знака #

cpi r18, 0x00 ; опущен ли флаг в r18? (по вводу # - 1, по вводу $ - 0)

brne PC+8 ; если поднят – переход к подпрограмме поиска $

ld r17, Y ; если опущен – то

cpi r17, 0b00100011 ; сравниваем значение r17 с бинарным кодом #

brne PC+3

ldi r18, 0x01 ; если совпадает – поднимаем флаг в r18 и выводим

out UDR, r17 ; значение r17 (то есть #)

inc YL ; сдвиг указателя по обработанным данным

reti

search\_0: ; подпрограмма распознавания конечного знака $

cpi r18, 0x01 ; поднят ли флаг в r18? (по вводу # - 1, по вводу $ - 0)

brne PC+8 ; если опущен – переход к инкрементированию Y

ld r17, Y

cpi r17, 0b00100100 ; значение в r17 это $?

brne PC+9 ; если нет–переход к подпрограмме распознавания 1й команды

ldi r18, 0x00 ; если да – опускаем флаг в r18

ldi r24, 0b00000111 ; записываем в r24 код, который выключает 3 диода

out PORTB, r24

out UDR, r17 ; вывод значения r17

inc YL ; сдвиг указателя по обработанным данным

ldi XH, high(arr\_1) ; указатель X теперь будет указывать на 1й массив

ldi XL, low(arr\_1)

reti

search\_1: ; основная подпрограмма распознавания (+1я буква 1го массива)

ld r17, Y ; запись в r17 значения по указателю Y (данные с UART)

ld r19, X ; запись в r19 значения по указателю X (данные из массива)

sub r17, r19

cpi r17, 0 ; сравниваем r17 и r19, если не равны –то указатель X будет

brne PC+9 ; указывать на 2й массив, переходим ко 2й подпрограмме

cpi r19, 0b01000111 ; если равны – сравниваем значение в r19 с кодом

brne PC+3 ; 1й буквы команды в 1м массиве, а если они совпадают-

ldi r24, 0b00000110 ; то включаем первый диод, и в любом случае

out PORTB, r24 ; выводим букву

out UDR, r19

inc YL ; сдвиг указателя по обработанным данным

inc XL ; переход к следующей букве массива

reti

ldi XH, high(arr\_2)

ldi XL, low(arr\_2)

search\_2: ; подпрограмма распознавания 1й буквы 2го массива

ld r17, Y ; запись в r17 значения по указателю Y (данные с UART)

ld r19, X ; запись в r19 значения по указателю X (данные из 2 массива)

sub r17, r19 ; сравнение r17 и r19, если не равны - то указатель X будет

cpi r17, 0 ; указывать на 3й массив, переходим к 3й подпрограмме,

brne PC+7 ; а если равны – включаем 2й диод и выводим букву

ldi r24, 0b00000101

out PORTB, r24

out UDR, r19

inc YL ; сдвиг указателя по обработанным данным

inc XL ; переход к следующему элементу 2го массива

reti

ldi XH, high(arr\_3)

ldi XL, low(arr\_3)

search\_3:

ld r17, Y ; запись в r17 значения по указателю Y (данные с UART)

ld r19, X ; запись в r19 значения по указателю X (данные из 3 массива)

sub r17, r19 ; сравнение r17 и r19, если не равны - то переходим к

cpi r17, 0 ; следующему записанному с UART символу, переходим к

brne PC+4 ; следующему элементу 3го массива, выходим из прерывания

ldi r24, 0b00000011

out PORTB, r24

out UDR, r19

inc YL ; сдвиг указателя по обработанным данным

inc XL

reti

start:

ldi r16, low(RAMEND) ; инициализация стека

out SPL, r16

ldi r16, high(RAMEND)

out SPH, r16

ldi r16, 0b00000111

out DDRB, r16 ; установка PB0, PB1 и PB2 в режим вывода

ldi r16, 0b00000111

out PORTB, r16 ; выключаем все диоды

ldi r16, 0x0C

out UBRRL, r16 ; установка скорости передачи данных

ldi r16, 0x98

out UCSRB, r16 ; разрешение передачи и приема данных, прерываний по приему данных

ldi r16, 0x86

out UCSRC, r16 ; установка размера посылки: 8 бит

sei ; разрешаем глобальные прерывания

ldi XH, high(arr\_1) ; указатель Х указывает на 1й массив

ldi XL, low(arr\_1)

ldi r16, 0b01000111 ; запись в 1й массив команды GO

st X+, r16

ldi r16, 0b01001111

st X+, r16

ldi XH, high(arr\_2) ; указатель X указывает на 2й массив

ldi XL, low(arr\_2)

ldi r16, 0b01001000 ; запись во 2й массив команды HELP

st X+, r16

ldi r16, 0b01000101

st X+, r16

ldi r16, 0b01001100

st X+, r16

ldi r16, 0b01010000

st X+, r16

ldi XH, high(arr\_3) ; указатель указывает на 3й массив

ldi XL, low(arr\_3)

ldi r16, 0b01010011 ; запись в 3й массив команды STOP

st X+, r16

ldi r16, 0b01010100

st X+, r16

ldi r16, 0b01001111

st X+, r16

ldi r16, 0b01010000

st X+, r16

ldi ZH, high(arr\_z) ; указатель Z указывает на кольцевой буфер,

ldi ZL, low(arr\_z) ; сдвигается по мере сохранения данных

ldi YH, high(arr\_z) ; указатель Y указывает на кольцевой буфер

ldi YL, low(arr\_z) ; сдвигается по мере обработки сохраненных данных

ldi XH, high(arr\_1) ; указатель X указывает на 1й массив с 1й командой

ldi XL, low(arr\_1) ; используется для перехода между 1, 2 и 3 массивами

ldi r18, 0x00 ; опускаем флаг в r18

forever:

cpi YL, 0x7A ;проверка на переполнение буфера, если переполнен -

brne PC+2 ;запись в указатель адреса 0го элемента кольц. буфера

ldi YL, 0x6A

cpi ZL, 0x7A

brne PC+2

ldi ZL, 0x6A

rjmp forever

**Приложение 2. Код программы (файл .prj).**

.MICRO "ATmega8"

.PROGRAM "curs.asm"

.TARGET "cursovaya\_butkovskiy.hex"

.TRACE

.POWER VDD=5 VSS=0

.CLOCK 2meg

.STORE 250m

X1 TTY(9600 8 0 0 1 1) PD0 PD1

D1 VDD D1\_node

R1 D1\_node PB0 330

D2 VDD D2\_node

R2 D2\_node PB1 330

D3 VDD D3\_node

R3 D3\_node PB2 330