1 Формулировка задания

Модуль проверки ПИН-кода. Программа должна предоставлять возможность по проверке введённого ПИН-кода на соответствие заданному. ПИН-код должен храниться в EEPROM и считываться при запуске МК (запись ПИН-кода в EEPROM следует выполнять с помощью AVRFlash). Для отображения проверяемого ПИН-кода используется блок из четырёх семисегментных индикаторов. Для ввода цифры ПИНкода используются кнопки, подключённые к PORTA и PORTB: PB0 – цифра 0, PB1 – 1, PB2 – 2, PB3 – 3, PB4 – 4, PB5 – 5, PB6 – 6, PB7 – 7, PA4 – 8, PA5 – 9. При вводе неправильного ПИН-кода загорается светодиод PA6 на 20 секунд, после чего программа возвращается к началу ввода ПИН-кода для проверки. При вводе неправильного ПИН-кода три раза подряд загораются светодиод PA7 и PA6, а программа перестаёт работать. При вводе правильного ПИН-кода загорается светодиод PA7, вернуться к началу режима проверки ПИН-кода (завершить текущий сеанс) можно с помощью прерывания INT1 (кнопка PD3) – выполнится очистка семисегментных индикаторов и погаснет светодиод PA7. Ввод четырёхзначного ПИН-кода осуществляется поразрядно, если в течение 7 секунд с момента ввода 1-3 цифры ПИН-кода не была введена следующая цифра программа возвращается к началу ввода ПИНкода для проверки. Ввод ПИН-кода осуществляется от младшего разряда к старшему. В начале работы программы на семисегментном индикаторе нет цифр, сразу после ввода каждой цифры она дописывается на семисегментный индикатор.

2 Схема лабораторной установки

Схема лабораторной установки показана на Рисунок 1.

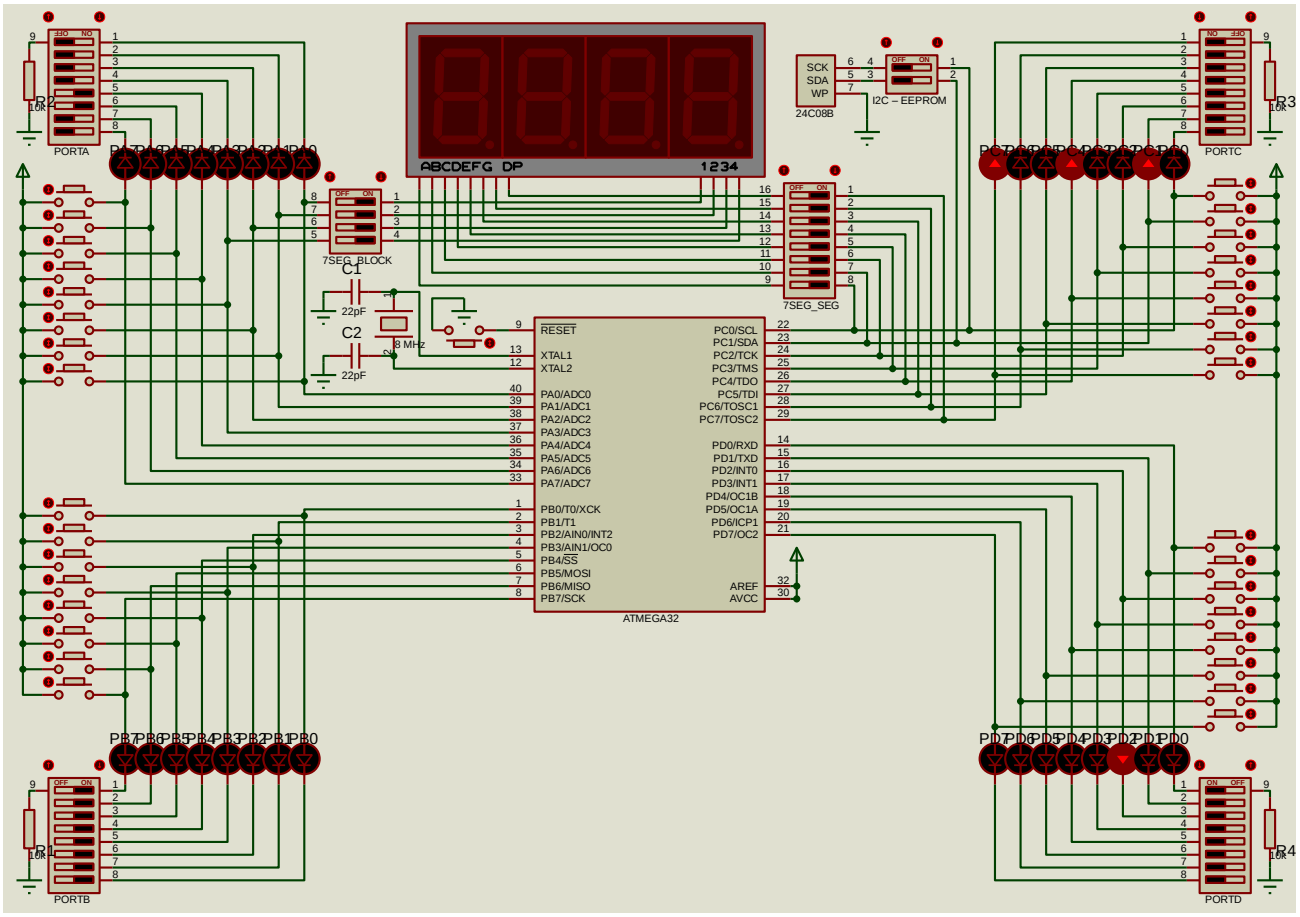

Рисунок 1 – Схема лабораторной установки

3 Блок-схема алгоритма работы программы

Блок-схема алгоритма работы программы показана на Рисунок 2. Блок-схемы подпрограмм, используемых в алгоритме, показаны на Рисунок 3-6.

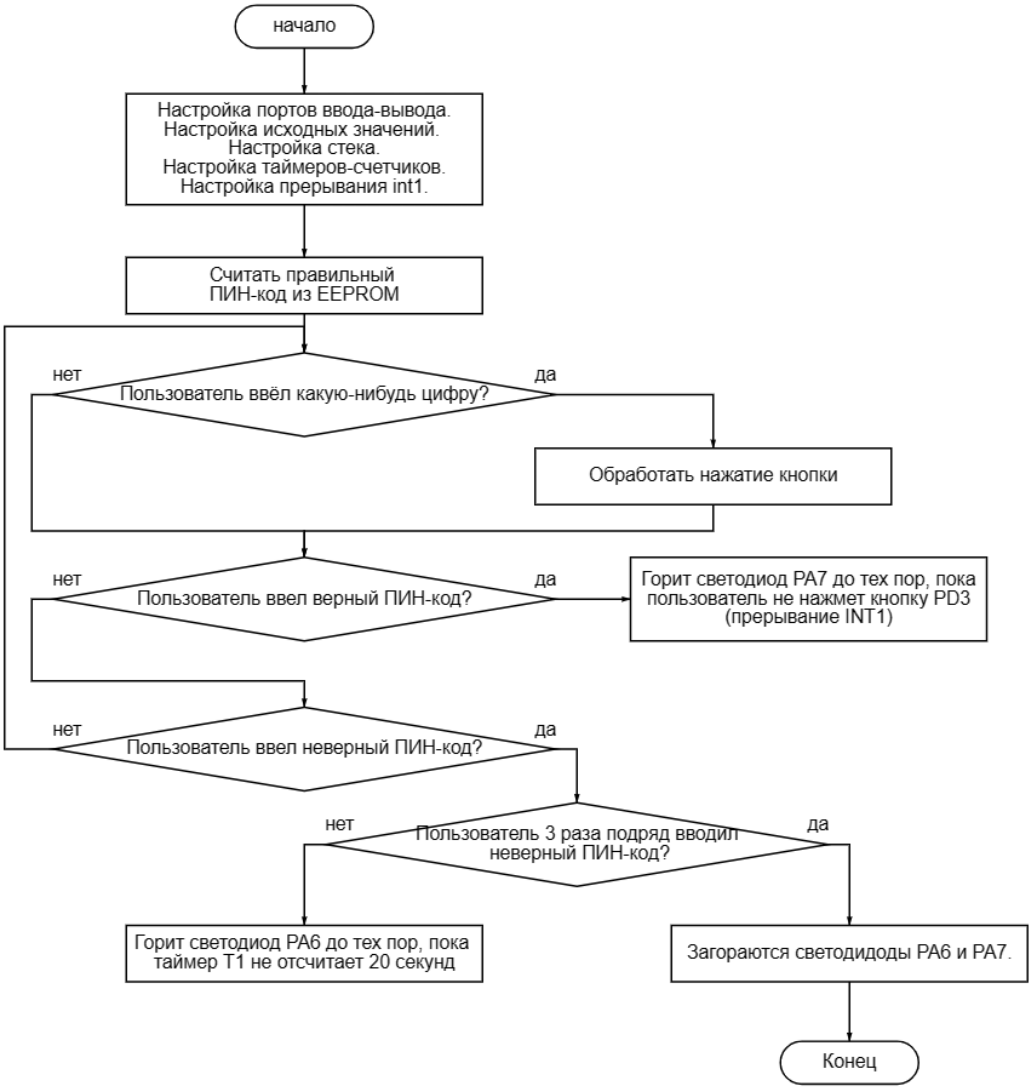

Рисунок 2 – Блок-схема алгоритма работы программы

Рисунок 3 – Блок-схема подпрограммы обработки прерывания INT1

Рисунок 4 – Блок-схема подпрограммы обработки прерывания TIMER0_OVF

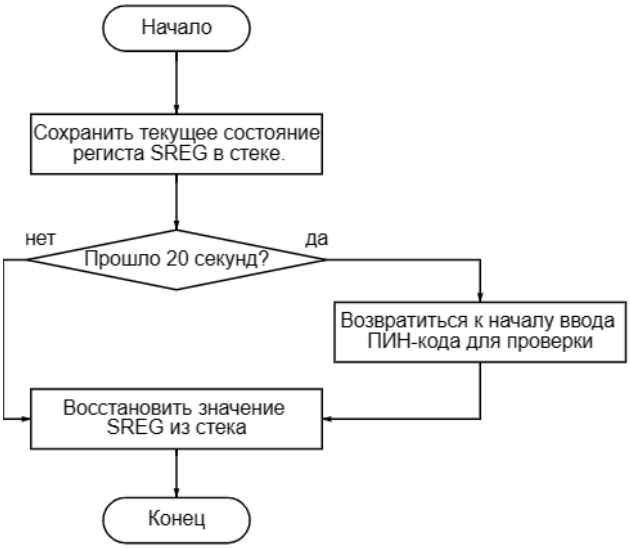

Рисунок 5 – Блок-схема подпрограммы обработки прерывания TIMER1_OVF

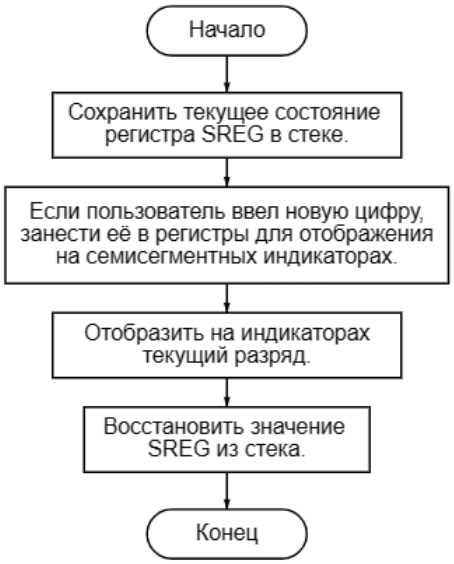

Рисунок 6 – Блок-схема подпрограммы обработки прерывания TIMER2_OVF

4 Ответы на контрольные вопросы

1. Посредством каких регистров производится конфигурирование таймера-счётчика?

Это можно показать на примере конфигурирования таймера-счётчика T0.

За конфигурацию таймера-счётчика T0 отвечает регистр TCCR0, он определяет источник тактирования таймера, коэффициент предделителя, режим работы таймера-счётчика T0 и поведение вывода OC0.

Названия бит регистра TCCR0 показаны на Рисунок 7.

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 7 – Регистр TCCR0

Биты CS02, CS01, CS00 (Clock Select) – определяют источник тактовой частоты для таймера T0 и задают коэффициент предделителя.

Биты WGM01, WGM00 (Wave Generator Mode) – определяют режим работы таймера-счётчика T0.

Биты COM01, COM00 (Compare Match Output Mode) – определяют поведение вывода OC0.

Бит регистра TCCR0 – это бит FOC0 (Force Output Compare). Этот бит предназначен для принудительного изменения состояния вывода OC0.

Регистр TCNT0 – восьмиразрядный счётный регистр. Когда таймер работает, по каждому импульсу тактового сигнала значение TCNT0 изменяется на единицу. В зависимости от режима работы таймера счётный регистр может или увеличиваться, или уменьшаться. Регистр TCNT0 можно как читать, так и записывать.

Регистр OCR0 – восьмиразрядный регистр сравнения. Его значение постоянно сравнивается со счётным регистром TCNT0, и в случае совпадения таймер может выполнять какие-то действия – вызывать прерывание, менять

состояние вывода OC0 и т.д. в зависимости от режима работы. Значение OCR0 можно как читать, так и записывать.

2. Какие источники импульсов могут применяться для увеличения таймера-счётчика и для каких целей?

Таймер/Счетчик считает либо тактовые импульсы от встроенного тактового генератора, либо со счетного входа.

3. В каких режимах могут работать таймеры-счётчики?

Всего существует 4 режима работы таймера-счетчика – нормальный режим (Normal), сброс таймера при совпадении (CTC), и два режима широтно-импульсной модуляции (FastPWM и Phase Correct PWM).

4. Как рассчитать начальное значение таймера-счётчика по заданному времени, которое должен отмерить таймер-счётчик до своего переполнения?

Для того чтобы рассчитать начальное значение таймера-счетчика по заданному времени, которое он должен отмерить до своего переполнения нужно:

1. Вычислить период одного такта таймера: $T = \frac{k}{F_{cpu}}$, где k – значение предделителя, F_{cpu} – частота работы микроконтроллера.
2. Вычислить требуемое количество тактов для заданного интервала времени t: $n = \frac{t}{T}$.
3. Вычислить начальное значение для счетного регистра по формуле: $256 - n$ для 8-битных таймеров-счетчиков (T0, T2) или $65536 - n$ для 16-битных (T1).

5. В чём состоит отличие работы таймера-счётчика в режиме таймера и в режиме счётчика?

Когда таймер-счетчик работает в режиме таймера он может тикать с разной частотой, тем самым подсчитывая время. Когда таймер работает в режиме счетчика он считать входящие извне импульсы.

5 Выводы

В ходе лабораторной работы были изучены принципы работы с таймерами-счётчиками, реализована программа, использующая обработку прерываний по переполнению таймеров-счётчиков и механизм прерываний для организации ввода.

ПРИЛОЖЕНИЕ А

Листинг программы на языке ассемблера

```
; начало программы
.org $000
     jmp RESET

; Внешнее прерывание и прерывания для таймеров
.org $004
     jmp EXT_INT1
.org $00A
     jmp TIM2_OVF
.org $012
     jmp TIMER1_OVF
.org $016
     jmp TIM0_OVF

; Цифры корректного кода (1-4)
.def CORRECT_CODE_1 = R1
.def CORRECT_CODE_2 = R2
.def CORRECT_CODE_3 = R3
.def CORRECT_CODE_4 = R4

; Цифры ввода (1-4)
.def USER_CODE_1 = R5
.def USER_CODE_2 = R6
.def USER_CODE_3 = R7
.def USER_CODE_4 = R8

; Текущая крайняя цифра ввода
.def CURR_CODE = R9

; Флаг попыток:
; 0x00 -  начальное значение, попыток не было
; 0xFF - корректный код
; 0x01 - некорректный код
```

```
.def CORRECT_FLAG = R10

; Текущее количество ошибок
.def MISSED = R11

; Текущее количество введённых цифр
.def CODE_COUNTER = R16

.def TMP = R17

; Для отсчёта времени таймерами 0 и 1 соответственно
.def T0 = R18
.def T1 = R19

; Вывод цифр на семисегментные индикаторы таймером 2
.def T2 = R20

; Коды цифр: крайней цифры ввода и всех введённых пользователем (1-4)
.def CURR_NUM = R21
.def NUM1 = R22
.def NUM2 = R23
.def NUM3 = R24
.def NUM4 = R25


;___________Начало программы____________;
RESET:
    ; Настройка портов:
    ; PA4, PA5 на ввод, оставшиеся PAx на вывод
    ldi TMP, 0xCF ; 11001111
    out DDRA, TMP
    clr TMP
    ; PBx все на ввод
    out DDRB, TMP
    ser TMP
```

```
    ; PCx все на вывод
    out DDRC, TMP
    ; PD3 на ввод, оставшиеся PDx на вывод
    ldi TMP, 0xF7 ; 11110111
    out DDRD, TMP

    ; Настройка таймеров-счётчиков:
    ; T0: предделитель 1024, режим Normal
    ldi TMP, 0b00000101
    out TCCR0, TMP
    clr TMP
    ; Счётный регистр в 0
    out TCNT0, TMP
    ; T2: предделитель 64, режим Normal
    ldi TMP, 0b00000011
    out TCCR2, TMP
    clr TMP
    ; Счётный регистр в 0
    out TCNT2, TMP
    ; T1 (B): предделитель 64, режим Normal
    ldi TMP, 0b00000011
    out TCCR1B, TMP
    clr TMP
    ; Счётный регистр в 0
    out TCNT1H, TMP
    out TCNT1L, TMP

    ; Настройка прерываний счётчиков:
    ; TOIE2 (переполнение на T2) и TOIE0 (переполнение на T0)
установлены
    ldi TMP, 0b01000001
    out TIMSK, TMP


    ser CURR_NUM
    ser NUM1
```

```
    ser NUM2
    ser NUM3
    ser NUM4
    mov USER_CODE_1, CURR_NUM
    mov USER_CODE_2, CURR_NUM
    mov USER_CODE_3, CURR_NUM
    mov USER_CODE_4, CURR_NUM
    clr TMP
    clr T0
    clr T1
    clr T2
    clr CODE_COUNTER
    clr CORRECT_FLAG
    clr MISSED

    ; Установка вершины стека в конец ОЗУ
    ldi TMP, HIGH(RAMEND) ; Старшие разряды адреса
    out SPH, TMP
    ldi TMP, LOW(RAMEND) ; Младшие разряды адреса
    out SPL, TMP

    ; Настройка условий прерываний:
    ; INT1 на восходящий фронт (0->1)
    ldi TMP, 0x0C ; 00001100
    out MCUCR, TMP
    ; Включено внешнее прерывание INT1
    ldi TMP, 0x80 ; 10000000
    out GICR, TMP
    out GIFR, TMP
    ; Глобальное разрешение прерываний
    sei

;___________Чтение корректного PIN из EEPROM____________;
read_pincode:
    ldi R26, 0
    ldi R27, 0
```

```
	rcall EERead
	mov CORRECT_CODE_1, R28
	ldi R26, 1
	rcall EERead
	mov CORRECT_CODE_2, R28
	ldi R26, 2
	rcall EERead
	mov CORRECT_CODE_3, R28
	ldi R26, 3
	rcall EERead
	mov CORRECT_CODE_4, R28

;___________Сеанс ввода пароля____________;
main:
	; Гасим PA6 и PA7
	cbi PORTA, 6
	cbi PORTA, 7
	; Читаем цифру
	sbic PINB, 0
	jmp read_0
	sbic PINB, 1
	jmp read_1
	sbic PINB, 2
	jmp read_2
	sbic PINB, 3
	jmp read_3
	sbic PINB, 4
	jmp read_4
	sbic PINB, 5
	jmp read_5
	sbic PINB, 6
	jmp read_6
	sbic PINB, 7
	jmp read_7
	sbic PINA, 4
	jmp read_8
```

```
    sbic PINA, 5
    jmp read_9
    ; Проверка количества введенных цифр: проверка или ожидание следующей
    cpi CODE_COUNTER, 0x04
    breq check_pin
    jmp main

;___________Проверка пароля____________;
check_pin:
    ; Подсчёт количества ошибок
    ldi TMP, 0x00
    cpse USER_CODE_1, CORRECT_CODE_1
    inc TMP
    cpse USER_CODE_2, CORRECT_CODE_2
    inc TMP
    cpse USER_CODE_3, CORRECT_CODE_3
    inc TMP
    cpse USER_CODE_4, CORRECT_CODE_4
    inc TMP
    cpi TMP, 0x00
    ; Обработка корректного кода
    breq found_correct_pin
    ; Обработка ошибки
    inc MISSED
    ldi TMP, 0x01
    mov CORRECT_FLAG, TMP
    ; Глобальный запрет прерываний, чтобы переключить T1 и T2 на прерывание переполнения (TOIE2, TOIE1), отключить прерывания T0
    cli
    ldi TMP, 0b01000100
    out TIMSK, TMP
    sei
    jmp incorrect_pin

;___________Обработка корректного кода____________;
```

```
found_correct_pin:
    ldi TMP, 0xFF
    mov CORRECT_FLAG, TMP
    ; Глобальный запрет прерываний, чтобы переключить T2 на прерывание переполнения (TOIE2), отключить прерывания T0, T1
    cli
    ldi TMP, 0b01000000
    out TIMSK, TMP
    sei
    jmp correct_pin

correct_pin:
    ; Зажечь PA7 и остаться в этом цикле, пока не будет нажата PD3 (вызов INT1)
    sbi PORTA, 7
    ldi TMP, 0xFF
    cpse CORRECT_FLAG, TMP
    jmp main
    jmp correct_pin

;___________Обработка некорректного кода____________;
incorrect_pin:
    ; Если число ошибок достигло границы, выходим на завершение программы
    ldi TMP, 0x03
    cp MISSED, TMP
    breq attempts3
    sbi PORTA, 6
    ldi TMP, 0x01
    cpse CORRECT_FLAG, TMP
    jmp main
    jmp incorrect_pin

attempts3:
    ; Отключение прерываний, зажечь PA6, PA7, остаться в цикле
    cli
```

```
	clr TMP
	out PORTA, TMP
	out PORTC, TMP
	sbi PORTA, 6
	sbi PORTA, 7
	jmp attempts3

;___________Вспомогательное для ввода____________;
; Пока зажаты кнопки, не происходит повторного ввода, ожидаем, когда нужный бит в PINX станет равным нулю
stop_readingB:
	in TMP, PINB
	cpi TMP, 0x00
	brne stop_readingB
	ret

stop_readingA8:
	sbic PINA, 4
	jmp stop_readingA8
	ret

stop_readingA9:
	sbic PINA, 5
	jmp stop_readingA9
	ret

; Чтение чисел:
read_0:
	ldi CURR_NUM, 0x3F
	ldi TMP, 0x00
	mov CURR_CODE, TMP
	inc CODE_COUNTER
	ldi T0, 0x00
	call stop_readingB
	jmp main
```

```
read_1:
     ldi CURR_NUM, 0x06
     ldi TMP, 0x01
     mov CURR_CODE, TMP
     inc CODE_COUNTER
     ldi T0, 0x00
     call stop_readingB
     jmp main

read_2:
     ldi CURR_NUM, 0x5B
     ldi TMP, 0x02
     mov CURR_CODE, TMP
     inc CODE_COUNTER
     ldi T0, 0x00
     call stop_readingB
     jmp main

read_3:
     ldi CURR_NUM, 0x4F
     ldi TMP, 0x03
     mov CURR_CODE, TMP
     inc CODE_COUNTER
     ldi T0, 0x00
     call stop_readingB
     jmp main

read_4:
     ldi CURR_NUM, 0x66
     ldi TMP, 0x04
     mov CURR_CODE, TMP
     inc CODE_COUNTER
     ldi T0, 0x00
     call stop_readingB
     jmp main
```

```
read_5:
     ldi CURR_NUM, 0x6D
     ldi TMP, 0x05
     mov CURR_CODE, TMP
     inc CODE_COUNTER
     ldi T0, 0x00
     call stop_readingB
     jmp main

read_6:
     ldi CURR_NUM, 0x7D
     ldi TMP, 0x06
     mov CURR_CODE, TMP
     inc CODE_COUNTER
     ldi T0, 0x00
     call stop_readingB
     jmp main

read_7:
     ldi CURR_NUM, 0x07
     ldi TMP, 0x07
     mov CURR_CODE, TMP
     inc CODE_COUNTER
     ldi T0, 0x00
     call stop_readingB
     jmp main

read_8:
     ldi CURR_NUM, 0x7F
     ldi TMP, 0x08
     mov CURR_CODE, TMP
     inc CODE_COUNTER
     ldi T0, 0x00
     call stop_readingA8
     jmp main
```

```
read_9:
     ldi CURR_NUM, 0x6F
     ldi TMP, 0x09
     mov CURR_CODE, TMP
     inc CODE_COUNTER
     ldi T0, 0x00
     call stop_readingA9
     jmp main

;___________Таймеры___________;
; Сохранять SREG перед тем, как работать с прерыванием
; Обработка прерывания по переполнению T0
TIM0_OVF:
     in R28, SREG
     push R28
     inc T0
     cpi T0, 213
     breq passed_7_seconds
     pop R28
     out SREG, R28
     reti

; После ожидания нужно подготовить всё к следующей итерации
passed_7_seconds:
     clr R28
     out PORTA, R28
     ser CURR_NUM
     ser NUM1
     ser NUM2
     ser NUM3
     ser NUM4
     mov USER_CODE_1, CURR_NUM
     mov USER_CODE_2, CURR_NUM
     mov USER_CODE_3, CURR_NUM
     mov USER_CODE_4, CURR_NUM
     clr TMP
```

```
    clr T0
    out TCNT0, T0
    clr T1
    out TCNT1H, T1
    out TCNT1L, T1
    clr T2
    out TCNT2, T2
    clr CODE_COUNTER
    clr CORRECT_FLAG
    pop R28
    out SREG, R28
    reti

; Обработка прерывания по переполнению T1
TIMER1_OVF:
    in R30, SREG
    push R30
    inc T1
    cpi T1, 38
    breq passed_20_seconds
    pop R30
    out SREG, R30
    reti

; После ожидания нужно подготовить всё к следующей итерации
passed_20_seconds:
    clr R30
    out PORTA, R30
    ser CURR_NUM
    ser NUM1
    ser NUM2
    ser NUM3
    ser NUM4
    mov USER_CODE_1, CURR_NUM
    mov USER_CODE_2, CURR_NUM
    mov USER_CODE_3, CURR_NUM
```

```
	mov USER_CODE_4, CURR_NUM
	clr TMP
	clr T0
	out TCNT0, T0
	clr T1
	out TCNT1H, T1
	out TCNT1L, T1
	clr T2
	out TCNT2, T2
	clr CODE_COUNTER
	clr CORRECT_FLAG
	ldi R30, 0b01000001
	out TIMSK, R30
	pop R30
	out SREG, R30
	reti

; Переполнение T2 используется для вывода цифр на семисегментные индикаторы
TIM2_OVF:
	in R29, SREG
	push R29
	;clr R29
	;out PORTA, R29 ; гасим все разряды
	cpi CODE_COUNTER, 0x00
	breq end_print
	cpi CODE_COUNTER, 0x01
	breq change_num1
	cpi CODE_COUNTER, 0x02
	breq change_num2
	cpi CODE_COUNTER, 0x03
	breq change_num3
	cpi CODE_COUNTER, 0x04
	breq change_num4
rr1:
	cpi T2, 0x00
```

```
     breq print_razr1
rr2:
     cpi T2, 0x01
     breq print_razr2
rr3:
     cpi T2, 0x02
     breq print_razr3
rr4:
     cpi T2, 0x03
     breq print_razr4
end_rr:
     inc T2
     sbrc T2, 3
     clr T2
end_print:
     pop R29
     out SREG, R29
     reti

change_num1:
     mov NUM1, CURR_NUM
     mov USER_CODE_1, CURR_CODE
     jmp rr1

change_num2:
     mov NUM2, CURR_NUM
     mov USER_CODE_2, CURR_CODE
     jmp rr1

change_num3:
     mov NUM3, CURR_NUM
     mov USER_CODE_3, CURR_CODE
     jmp rr1

change_num4:
     mov NUM4, CURR_NUM
```

```
     mov USER_CODE_4, CURR_CODE
     jmp rr1

; Вывод каждого из разрядов (1-4)
print_razr1:
     cpi CODE_COUNTER, 0x01
     brsh out_razr1
     jmp end_rr
out_razr1:
     ldi R29, 0b00000001
     out PORTA, R29
     out PORTC, NUM1
     jmp rr2

print_razr2:
     cpi CODE_COUNTER, 0x02
     brsh out_razr2
     jmp end_rr
out_razr2:
     ldi R29, 0b00000010
     out PORTA, R29
     out PORTC, NUM2
     jmp rr3

print_razr3:
     cpi CODE_COUNTER, 0x03
     brsh out_razr3
     jmp end_rr
out_razr3:
     ldi R29, 0b00000100
     out PORTA, R29
     out PORTC, NUM3
     jmp rr4

print_razr4:
     cpi CODE_COUNTER, 0x04
```

```
     breq out_razr4
     jmp end_rr
out_razr4:
     ldi R29, 0b00001000
     out PORTA, R29
     out PORTC, NUM4
     jmp end_rr

; Обработка INT1: очистка временных хранилищ (пользовательский код и т.д.), флагов, таймеров
EXT_INT1:
     in R27, SREG
     push R27
     clr R27
     out PORTA, R27
     cbi PORTA, 7
     ser CURR_NUM
     ser NUM1
     ser NUM2
     ser NUM3
     ser NUM4
     mov USER_CODE_1, CURR_NUM ;
     mov USER_CODE_2, CURR_NUM
     mov USER_CODE_3, CURR_NUM
     mov USER_CODE_4, CURR_NUM
     clr TMP
     clr T0
     out TCNT0, T0
     clr T1
     out TCNT1H, T1
     out TCNT1L, T1
     clr T2
     out TCNT2, T2
     clr CODE_COUNTER
     clr CORRECT_FLAG
     clr MISSED
```

```
    ldi R27, 0b01000001
    out TIMSK, R27
    pop R27
    out SREG, R27
    reti

; Чтение из памяти: ожидание окончания другого чтения, если оно есть, после чего из R26, R27 смотрим нужный адрес
EERead:
    sbic EECR,EEWE
    rjmp EERead
    out  EEARL, R26
    out  EEARH, R27
    sbi  EECR,EERE
    in R28, EEDR
    ret
```