Министерство образования и науки Российской Федерации
Федеральное государственное унитарное предприятие
«Российское научно-производственное объединение
«Росучприбор»

ЗАО НПК «КОМПЬЮТЕРЛИНК»

ОКП

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

"АРХИТЕКТУРА, ПРОГРАММИРОВАНИЕ И ПРИМЕНЕНИЕ 8-РАЗРЯДНЫХ МИКРОКОНТРОЛЛЕРОВ семейства ATMEL ATmega"

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В цикле работ данного практикума студенты знакомятся с архитектурой 8-разрядного микроконтроллера AVR mega128, изучают систему его команд и методы адресации, осваивают интегрированную систему программирования, получают практические навыки программирования микроконтроллерных систем на языке Ассемблер.

Микроконтроллер ATmega128 является старшей моделью семейства ATmega фирмы Atmel. Семейство AVR (AT) удачно воплощает современные тенденции архитектуры RISC микроконтроллеров, что в сочетании с достижениями фирмы Atmel в области создания Flash-памяти, сделало его весьма популярным на мировом рынке 8-разрядных микроконтроллеров.

Семейство AVR включает около двух десятков типов 8-разрядных микроконтроллеров трех основных линий:

- *Tiny AVR* представляют собой низкостоимостные микроконтроллеры в 8-выводном корпусе.
- *Classic AVR* являются устаревшей линией семейства. Быстродействие некоторых моделей достигает 16 MIPS, Flash ROM программ 2-8 Кбайт, EEPROM данных 64-512 байт, ОЗУ данных 128-512 байт;
- *Mega AVR* представляет собой основную модель, ориентированную на высокопроизводительную работу со сложными задачами, требующими больших ресурсов памяти. Flash ROM программ составляет 8-128 Кбайт, EEPROM данных 64-512 байт, ОЗУ данных 2-4 Кбайт. Имеются 10-разрядный АЦП и аналоговый компаратор.

В составе семейства существуют модификации с низковольтным питанием. Благодаря универсальности, широкому набору функциональных возможностей, высоким техническим характеристикам микроконтроллеры семейства AVR находят все более широкое применение в системах управления различными объектами.

Все микроконтроллеры семейства AVR имеют общие принципы функционирования, реализуют единую систему команд, используют одинаковые методы адресации. Изучаемый в данном практикуме микроконтроллер ATmega128 является типичным представителем семейства AVR. Знакомство с ним позволит освоить основные методы проектирования, программирования и применения систем управления на базе современных RISC микроконтроллеров.

1. СТРУКТУРА И ФУНКЦИОНИРОВАНИЕ МИКРОКОНТРОЛЛЕРА ATMEGA128

Структура микроконтроллера ATmega128 включает следующие функциональные блоки:
- 8-разрядное арифметическо-логическое устройство (АЛУ);
- внутреннюю flash-память программ объемом 128 Кбайт с возможностью внутрисистемного программирования через последовательный интерфейс;
- 32 регистра общего назначения;
- внутреннюю EEPROM память данных объемом 4 Кбайт;
- внутреннее ОЗУ данных объемом 4 Кбайт;
- 6 параллельных 8-разрядных портов;
- 4 программируемых таймера-счетчика;
- 10-разрядный 8-канальный АЦП и аналоговый компаратор;
- последовательные интерфейсы UART0, UART0, TWI и SPI;
- блоки прерывания и управления (включая сторожевой таймер).

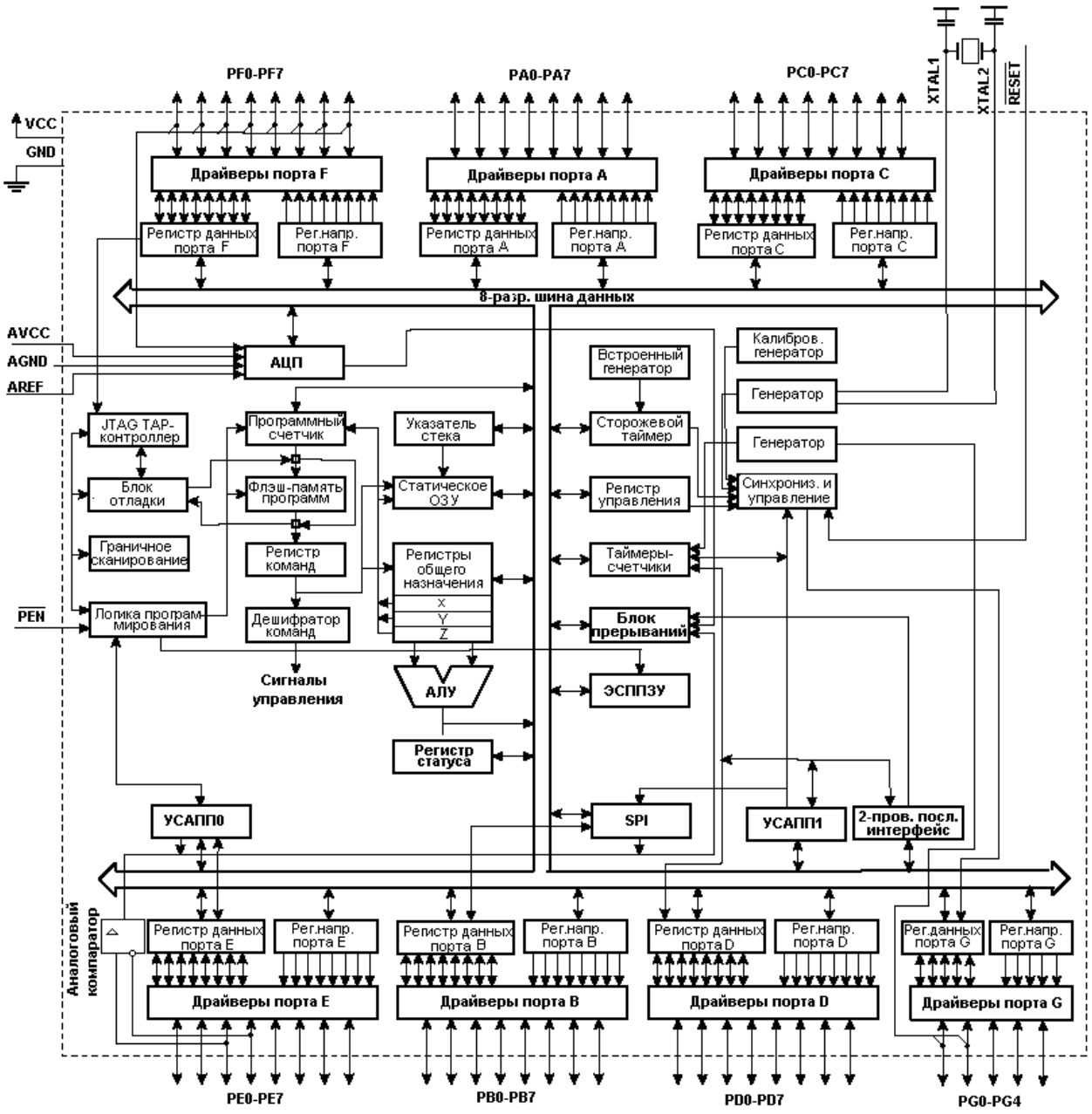


Рис.1.1. Структурная схема микроконтроллера.

На рис.1.2 изображен корпус и приведено назначение выводов микроконтроллера ATmega128. В скобках указана альтернативная функция вывода, если она существует.

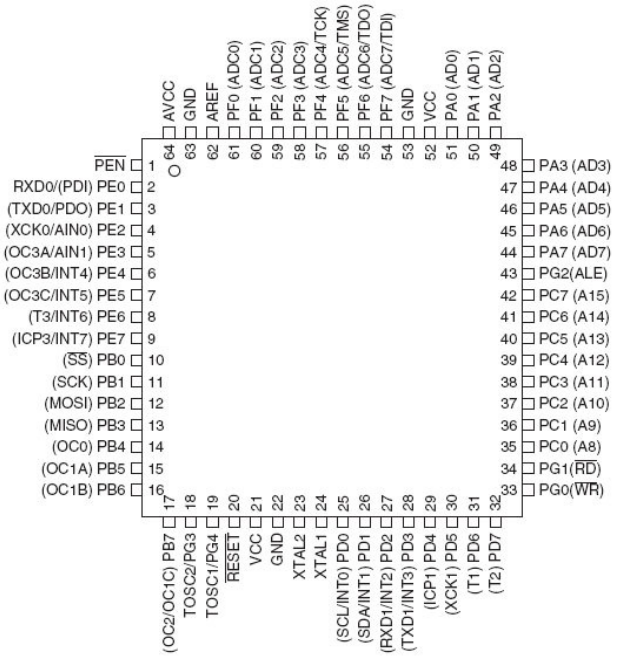


Рис.1.2 Вид корпуса и назначение выводов микроконтроллера ATmega128

Port A (PA7..PA). 8-разрядный двунаправленный порт. К выводам порта могут быть подключены встроенные нагрузочные резисторы (отдельно к каждому разряду). Выходные буферы обеспечивают ток 20 мА и способность прямо управлять светодиодным индикатором. При использовании выводов порта в качестве входов и установке внешнего сигнала в низкое состояние, ток будет вытекать только при подключенных встроенных нагрузочных резисторах. Порт А при наличии внешней памяти данных используется для организации мультиплексируемой шины адреса/данных.

Port B (PB7..PB0). 8-разрядный двунаправленный порт со встроенными нагрузочными резисторами. Выходные буферы обеспечивают ток 20 мА. При использовании выводов порта в качестве входов и установке внешнего сигнала в низкое состояние, ток будет вытекать только при подключенных встроенных нагрузочных резисторах. Порт В используется также при реализации специальных функций.

Port C (PC7..PC0). Порт С является 8-разрядным выходным портом. Выходные буферы обеспечивают ток 20 мА. Порт С при наличии внешней памяти данных используется для организации шины адреса.

Port D (PD7..PD0). 8-разрядный двунаправленный порт со встроенными нагрузочными резисторами. Выходные буферы обеспечивают ток 20 мА. При использовании выводов порта в качестве входов и установке внешнего сигнала в низкое состояние, ток будет вытекать только при подключенных встроенных нагрузочных резисторах. Порт D используется также при реализации специальных функций.

Port E (PE7..PE0). 8-разрядный двунаправленный порт со встроенными нагрузочными резисторами. Выходные буферы обеспечивают ток 20 мА. При использовании выводов порта в качестве входов и установке внешнего сигнала в низкое состояние, вытекающий через них ток обеспечивается только при подключенных встроенных нагрузочных резисторах. Порт Е используется также при реализации специальных функций.

Port F (PF7..PF0). 8-разрядный входной порт. Входы порта используются также как аналоговые входы аналого-цифрового преобразователя.

RESET. Вход сброса. Для выполнения сброса необходимо удерживать низкий уровень на входе более 50 нс.

XTAL1, XTAL2. Вход и выход инвертирующего усилителя генератора тактовой частоты.

TOSC1, TOSC2. Вход и выход инвертирующего усилителя генератора таймера/счетчика.

WR, RD. Стробы записи и чтения внешней памяти данных.
ALE. Строб разрешения фиксации адреса внешней памяти. Строб ALE используется для фиксации младшего байта адреса с выводов AD0-AD7 в защелке адреса в течение первого цикла обращения. В течение второго цикла обращения выводы AD0-AD7 используются для передачи данных.
AVCC. Напряжение питания аналого-цифрового преобразователя. Вывод подсоединяется к V_{CC} через низкочастотный фильтр.
AREF. Вход опорного напряжения для аналого-цифрового преобразователя. На этот вывод подается напряжение в диапазоне между AGND и AVCC.
AGND. Это вывод должен быть подсоединен к отдельной аналоговой земле, если она есть на плате. В ином случае вывод подсоединяется к общей земле.
PEN. Вывод разрешения программирования через последовательный интерфейс. При удержании сигнала на этом выводе на низком уровне после включения питания, прибор переходит в режим программирования по последовательному каналу.
V_{CC}, GND. Напряжение питания и земля

Микроконтроллеры AVR имеют раздельные пространства адресов памяти программ и данных (гарвардская архитектура). Организация памяти показана на рис. 1.3.

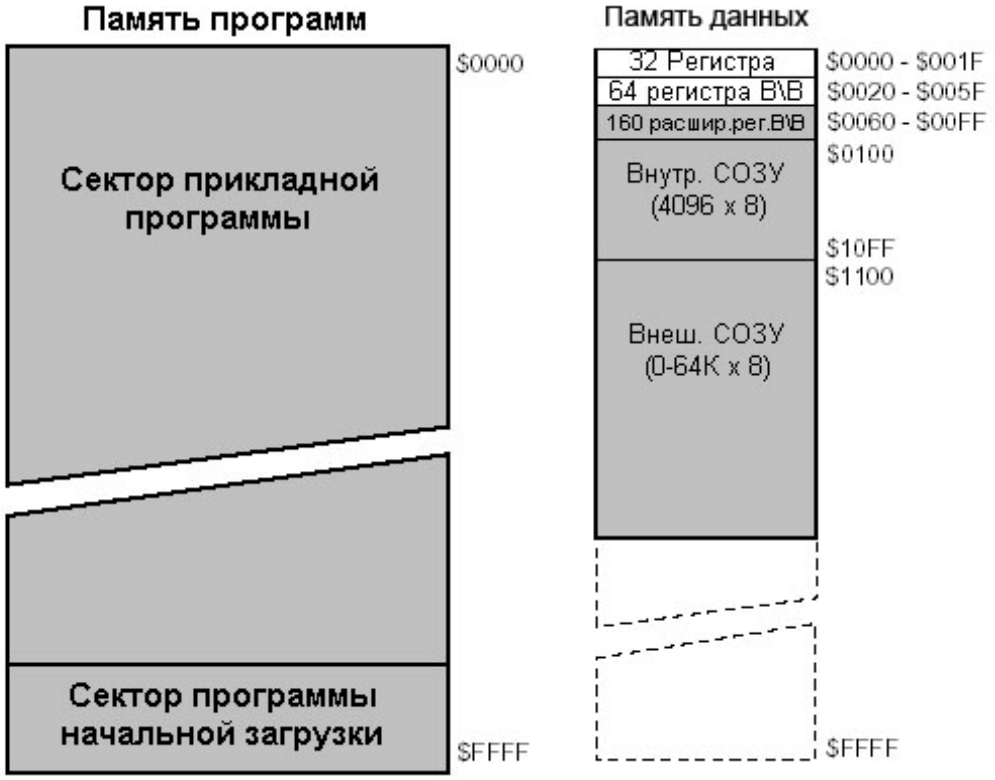


Рис.1.3. Организация памяти микроконтроллера ATmega128

Высокие характеристики семейства **AVR** обеспечиваются следующими особенностями архитектуры:

- В качестве памяти программ используется внутренняя flash-память. Она организована в виде матрицы 16-разрядных ячеек и может загружаться программатором, либо через порт SPI;
- Система команд включает 133 инструкций;
- 16-разрядные память программ и шина команд вместе с одноуровневым конвейером позволяют выполнить большинство инструкций за один такт синхрогенератора (50 нс при частоте F_{OSC}=20 МГц);
- память данных имеет 8-разрядную организацию. Младшие 32 адреса пространства занимают регистры общего назначения, далее следуют 64 адреса регистров ввода-

вывода, затем внутреннее ОЗУ данных объемом до 4096 ячеек. Возможно применение внешнего ОЗУ данных объемом до 60 Кбайт;

- внутренняя энергонезависимая память типа EEPROM объемом до 4 Кбайт представляет собой самостоятельную матрицу, обращение к которой осуществляется через специальные регистры ввода-вывода;

Как видно из рис.1.4, 32 регистра общего назначения включены в сквозное адресное пространство ОЗУ данных и занимают младшие адреса. Хотя физически регистры выделены из памяти данных, такая организация обеспечивает гибкость в работе. Файл регистров общего назначения прямо связан с АЛУ, каждый из регистров способен работать как аккумулятор. Большинство команд выполняются за один такт, при этом из регистров файла могут быть выбраны два операнда, выполнена операция и результат возвращен в регистровый файл. Старшие шесть регистров файла могут использоваться как три 16-разрядных регистра, и выполнять роль, например, указателей при косвенной адресации.

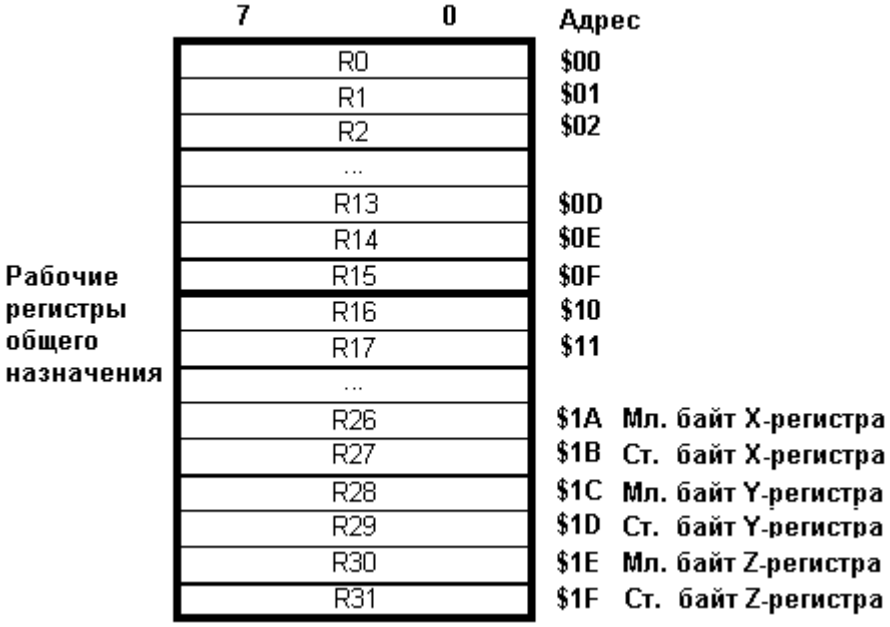


Рис.1.4. Регистры общего назначения микроконтроллера ATmega128

Следующие 64 адреса за регистрами общего назначения занимают регистры ввода-вывода. В этой области сгруппированы все регистры данных, управления и статуса внутренних программируемых блоков ввода-вывода. При использовании команд IN и OUT используются адреса ввода-вывода с $00 по $3F. Но к регистрам ввода-вывода можно обращаться и как к ячейкам внутреннего ОЗУ. При этом к непосредственному адресу ввода-вывода прибавляется $20. Адрес регистра как ячейки ОЗУ приводится далее в круглых скобках. Регистры ввода-вывода с $00 ($20) по $1F ($3F) имеют программно доступные биты. Обращение к ним осуществляется командами SBI и CBI, а проверка состояния – командами SBIS и SBIC. В таблице В1 приведен список регистров ввода-вывода.

Таблица 1.1. Регистры ввода-вывода микроконтроллера ATmega128

Название	Адрес	Функция
UCSR1C	($9D)	Регистр управления и состояния C USART1
UDR1	($9C)	Регистр данных USART1
UCSR1A	($9B)	Регистр управления и состояния A USART1
UCSR1B	($9A)	Регистр управления и состояния B USART1
UBRR1L	($99)	Регистр скорости передачи USART1, младший байт
UBRR1H	($98)	Регистр скорости передачи USART1, старший байт
UCSR0C	($95)	Регистр управления и состояния C USART0

UBRR0H	($90)	Регистр скорости передачи USART0, старший байт
TCCR3C	($8C)	Регистр управления C таймера/счетчика T3
TCCR3A	($8B)	Регистр управления A таймера/счетчика T3
TCCR3B	($8A)	Регистр управления B таймера/счетчика T3
TCNT3H	($89)	Счетный регистр таймера/счетчика T3, старший байт
TCNT3L	($88)	Счетный регистр таймера/счетчика T3, младший байт
OCR3AH	($87)	Регистр совпадения A таймера/счетчика T3, старший байт
OCR3AL	($86)	Регистр совпадения A таймера/счетчика T3, младший байт
OCR3BH	($85)	Регистр совпадения B таймера/счетчика T3, старший байт
OCR3BL	($84)	Регистр совпадения B таймера/счетчика T3, младший байт
OCR3CH	($83)	Регистр совпадения C таймера/счетчика T3, старший байт
OCR3CL	($82)	Регистр совпадения C таймера/счетчика T3, младший байт
ICR3H	($81)	Регистр захвата таймера/счетчика T3, старший байт
ICR3L	($80)	Регистр захвата таймера/счетчика T3, младший байт
ETIMSK	($7D)	Дополнительный регистр маски прерываний от таймеров/счетчиков
ETIFR	($7C)	Дополнительный регистр флагов прерываний от таймеров/счетчиков
TCCR1C	($7A)	Регистр управления C таймера/счетчика T1
OCR1CH	($79)	Регистр совпадения C таймера/счетчика T1, старший байт
OCR1CL	($78)	Регистр совпадения C таймера/счетчика T1, младший байт
TWCR	($74)	Регистр управления TWI
TWDR	($73)	Регистр данных TWI
TWAR	($72)	Регистр адреса TWI
TWSR	($71)	Регистр состояния TWI
TWBR	($70)	Регистр скорости передачи TWI
OSCCAL	($6F)	Регистр калибровки тактового генератора
XMCRA	($6D)	Регистр управления A внешней памятью
XMCRB	($6C)	Регистр управления B внешней памятью
EICRA	($6A)	Регистр управления A внешними прерываниями
SPMCR	($68)	Регистр управления памятью программ
PORTG	($65)	Регистр данных порта G
DDRG	($64)	Регистр направления данных порта G
PING	($63)	Выводы порта G
PORTF	($62)	Регистр данных порта F
DDRF	($61)	Регистр направления данных порта F
SREG	$3F($5F)	Регистр состояния
SPH	$3E($5E)	Указатель стека, старший байт
SPL	$3D($5D)	Указатель стека, младший байт
XDIV	$3C($5C)	Регистр управления делителем тактовой частоты
RAMPZ	$3B($5B)	Регистр выбора страницы
EICRB	$3A($5A)	Регистр управления B внешними прерываниями
EIMSK	$39($59)	Регистр маски внешних прерываний
EIFR	$38 ($58)	Регистр флагов внешних прерываний
TIMSK	$37($57)	Регистр маски прерываний от таймеров/счетчиков
TIFR	$36($56)	Регистр флагов прерываний от таймеров/счетчиков
MCUCR	$35($55)	Регистр управления микроконтроллером
MCUCSR	$34($54)	Регистр управления и состояния микроконтроллера
TCCR0	$33($53)	Регистр управления таймером/счетчиком T0
TCNT0	$32($52)	Счетный регистр таймера/счетчика T0
OCR0	$31($51)	Регистр совпадения таймера/счетчика T0
ASSR	$30($50)	Регистр состояния асинхронного режима
TCCR1A	$2F($4F)	Регистр управления A таймера/счетчика T1
TCCR1B	$2E($4E)	Регистр управления B таймера/счетчика T1
TCNT1H	$2D($4D)	Счетный регистр таймера/счетчика T1, старший байт
TCNT1L	$2C($4C)	Счетный регистр таймера/счетчика T1, младший байт
OCR1AH	$2B($4B)	Регистр совпадения A таймера/счетчика T1, старший байт

OCR1AL	$2A($4A)	Регистр совпадения A таймера/счетчика T1, младший байт
OCR1BH	$29($49)	Регистр совпадения B таймера/счетчика T1, старший байт
OCR1BL	$28($48)	Регистр совпадения B таймера/счетчика T1, младший байт
ICR1H	$27($47)	Регистр захвата таймера/счетчика T1, старший байт
ICR1L	$26($46)	Регистр захвата таймера/счетчика T1, младший байт
TCCR2	$25($45)	Счетный регистр таймера/счетчика T2
TCNT2	$24($44)	Регистр совпадения таймера/счетчика T2
OCR2	$23($43)	Регистр совпадения таймера/счетчика T2
OCDR	$22($42)	Регистр внутрисхемной отладки
WDTCR	$21($41)	Регистр управления сторожевым таймером
SFIOR	$20($40)	Регистр специальных функций
EEARH	$1F($3F)	Регистр адреса EEPROM, старший байт
EEARL	$1E($3E)	Регистр адреса EEPROM, младший байт
EEDR	$1D($3D)	Регистр данных EEPROM
EECR	$1C($3C)	Регистр управления EEPROM
PORTA	$1B($3B)	Регистр данных порта A
DDRA	$1A($3A)	Регистр направления данных порта A
PINA	$19($39)	Выводы порта A
PORTB	$18($38)	Регистр данных порта B
DDRB	$17($37)	Регистр направления данных порта B
PINB	$16($36)	Выводы порта B
PORTC	$15($35)	Регистр данных порта C
DDRC	$14($34)	Регистр направления данных порта C
PINC	$13($33)	Выводы порта C
PORTD	$12($32)	Регистр данных порта D
DDRD	$11($31)	Регистр направления данных порта D
PIND	$10($30)	Выводы порта D
SPDR	$0F($2F)	Регистр данных SPI
SPSR	$0E($2E)	Регистр состояния SPI
SPCR	$0D($2D)	Регистр управления SPI
UDR0	$0C($2C)	Регистр данных USART0
UCSR0A	$0B($2B)	Регистр управления и состояния A USART0
UCSR0B	$0A($2A)	Регистр управления и состояния B USART0
UBRR0L	$09($29)	Регистр скорости передачи USART0, младший байт
ACSR	$08($28)	Регистр управления и состояния аналогового компаратора
ADMUX	$07($27)	Регистр управления мультиплексором АЦП
ADCSRA	$06($26)	Регистр управления и состояния АЦП
ADCH	$05($25)	Регистр данных АЦП, старший байт
ADCL	$04($24)	Регистр данных АЦП, младший байт
PORTE	$03($23)	Регистр данных порта E
DDRE	$02($22)	Регистр направления данных порта E
PINE	$01($21)	Выводы порта E
PINF	$00($20)	Выводы порта F

В пространстве регистров ввода-вывода находятся и регистры управления процессором микроконтроллера: регистр состояния, указатель стека, регистр выбора страницы, регистр управления процессором, регистр управления коэффициентом деления частот. Формат этих регистров следующий:

Регистр состояния SREG.

	7	6	5	4	3	2	1	0	
$3F ($5F)	**I**	**T**	**H**	**S**	**V**	**N**	**Z**	**C**	**SREG**
Исходное значение	0	0	0	0	0	0	0	0	

Bit7 – Разрешение всех прерываний. Для разрешения прерываний этот бит должен быть установлен (=1). Разрешение конкретного прерывания выполняется регистрами маски прерывания EIMSK и TIMSK. Если этот бит очищен (=0), то ни одно из прерываний не обрабатывается. Бит аппаратно очищается после возникновения прерывания и устанавливается (разрешая последующие прерывания) командой RETI.

Bit 6 – Бит сохранения копии. Команды копирования бита BLD и BST используют этот бит как источник и приемник при операциях с битами. Командой BST бит регистра общего назначения копируется в бит T, командой BLD бит T копируется в бит регистра общего назначения.

Bit 5 – Флаг полупереноса. Флаг полупереноса указывает на перенос между тетрадами при выполнении ряда арифметических операций. Более подробная информация приведена в описании системы команд.

Bit 4 – Бит знака. Бит S имеет значение результата операции исключающее ИЛИ **(N⊕V)** над флагами отрицательного значения (N) и дополнения до двух флага переполнения (V) . Более подробная информация приведена в описании системы команд.

Bit 3 – Дополнение до двух флага переполнения. Этот бит поддерживает арифметику дополнения до двух. Более подробная информация приведена в описании системы команд.

Bit 2 – Флаг отрицательного значения. Этот флаг указывает на отрицательный результат ряда арифметических и логических операций. Более подробная информация приведена в описании системы команд.

Bit 1 – Флаг нулевого значения. Этот флаг указывает на нулевой результат ряда арифметических и логических операций. Более подробная информация приведена в описании системы команд.

Bit 0 – Флаг переноса. Этот флаг указывает на перенос при арифметических и логических операциях. Более подробная информация приведена в описании системы команд.

Указатель стека – SP.

	7	6	5	4	3	2	1	0	
$3E ($5E)	**SP15**	**SP14**	**SP13**	**SP12**	**SP11**	**SP10**	**SP9**	**SP8**	**SPH**
$3D ($5D)	**SP7**	**SP6**	**SP5**	**SP4**	**SP3**	**SP2**	**SP1**	**SP0**	**SPL**
Исходное	0	0	0	0	0	0	0	0	
значение	0	0	0	0	0	0	0	0	

Микроконтроллеры AVR оснащены 16-разрядным указателем стека, размещенным в двух регистрах ввода-вывода. Поскольку микроконтроллеры ATmega128 поддерживают объем ОЗУ до 64 Кбайт, то используются все 16 разрядов указателя стека.

Указатель стека указывает на область в ОЗУ данных, в которой размещается стек подпрограмм и прерываний. Начальный адрес указателя должен задаваться программно перед вызовом подпрограмм и разрешением прерываний. Начальное значение должно быть больше $60. Указатель стека декрементируется на единицу при каждом занесении командой PUSH данных в стек, и на две единицы при занесении в стек адреса при вызове подпрограммы или процедуры прерывания.

Указатель стека инкрементируется на единицу при извлечении данных из стека командой POP, и на две единицы при извлечении адреса из стека при возврате из подпрограммы (RET) или возврате из процедуры прерывания (RETI).

Регистр выбора страницы ОЗУ- RAMPZ.

	7	6	5	4	3	2	1	0	
$3B ($5B)	-	-	-	-	-	-	-	**RAMPZ0**	**RAMPZ**
Исходное значение	0	0	0	0	0	0	0	0	

Регистр RAMPZ используется обычно для определения страницы ОЗУ данных, к которой возможно обращение посредством указателя Z. Поскольку микроконтроллеры ATmega128 не поддерживают ОЗУ объемом более 64 К, этот регистр используется только для выбора страницы в памяти программ при использовании команды ELPM. Имеются следующие варианты использования единственного значащего бита RAMPZ0:

RAMPZ0 = 0: Команде ELPM доступна память программ с адресами от $0000 до $7FFF (младшие 64 Кбайт)

RAMPZ0 = 1: Команде ELPM доступна память программ с адресами от $8000 до $FFFF (старшие 64 Кбайт).

Отметим, что на команду LPM установки регистра RAMPZ не воздейсвуют.

Регистр управления MCU – MCUCR

	7	6	5	4	3	2	1	0	
$35 ($55)	**SRE**	**SRW**	**SE**	**SM1**	**SM0**	**-**	**-**	**-**	**MCUR**
Исходное значение	0	0	0	0	0	0	0	0	

Биты регистра управления MCU управляют выполнением основных функций MCU.

Bit 7 – Разрешение внешнего ОЗУ. Установленный (=1) бит SRE разрешает обращение к внешней памяти данных и переводит работу выводов AD0-7 (Порт A), A8-15 (Порт C), WR и RD на выполнение альтернативной функции. Бит SRE также перенастраивает установки направлений в соответствующих регистрах направления данных. Очистка бита SRE (=0) запрещает обращение к внешней памяти данных и восстанавливает нормальные установки направлений выводов и данных.

Bit 6 – Режим состояния ожидания. При установленном (=1) бите SRW к циклу обращений к внешней памяти данных добавляется один цикл ожидания. При сброшенном (=0) бите SRW обращение к внешней памяти выполняется по трехцикловой схеме.

Bit 5 – Разрешение режима Sleep. Установленный в 1 бит SE разрешает перевод MCU в режим sleep по команде SLEEP. Чтобы исключить перевод MCU в незапланированный режим sleep, рекомендуется устанавливать бит SE непосредственно перед выполнением команды SLEEP.

Bits 4,3 – Биты выбора режима Sleep. Эти биты позволяют выбрать один из трех возможных режимов энергосбережения, как показано в таблице B2.

Таблица 1.2. Выбор режима энергосбережения

SM1	SM0	Sleep Mode
0	0	Режим Idle
0	1	Зарезервирован
1	0	Режим Power Down
1	1	Режим Power Save

Bits 2..0 – Зарезервированные биты

Регистр управления делением частоты кварцевого генератора – XDIV

	7	6	5	4	3	2	1	0	
$3C ($5C)	**XDIVEN**	**XDIV6**	**XDIV5**	**XDIV4**	**XDIV3**	**XDIV2**	**XDIV1**	**XDIV0**	**XDIV**
Исходное значение	0	0	0	0	0	0	0	0	

Регистр XDIV используется для установки коэффициента деления частоты кварцевого генератора в диапазоне от 1 до 129. Эта возможность может быть использована для уменьшения энергопотребления.

Bit 7 – Разрешение деления частоты XTAL. При установленном (=1) бите XDIVEN тактовая частота CPU и всей периферии делится в соответствии с установленным битами XDIV6 - XDIV0 коэффициентом деления.

Bit 6..0 – Биты выбора коэффициента деления. Эти биты устанавливают коэффициент деления тактовой частоты при установленном бите XDIVEN. Если десятичное значение этих семи битов обозначить через d, то результирующая тактовая частота CPU будет определяться по формуле:

$$F_{CLK} = XTAL/(129-d)$$

Состояния этих битов можно изменить только при сброшенном бите XDIVEN. При установленном бите XDIVEN, записанное единовременно в биты XDIV6..XDIV0 значение будет определять коэффициент деления. При сбросе бита XDIVEN записанные в биты XDIV6..XDIV0 значения игнорируются. Поскольку делитель делит тактовую частоту поступающую на MCU, то и на периферийные устройства поступает тактовая частота с тем же коэффициентом деления.

За регистрами ввода-вывода следуют 4096 адресов внутреннего ОЗУ данных. При использовании внешней памяти данных адресуются оставшиеся 60 Кбайт. Таким образом, при использовании микросхем памяти объемом 64К будут потеряны 4Кбайт. При обращении к внешней и внутренней памяти данных используются одни и те же команды, но во втором случае сигналы RD# и WR# не активизируются.

Работа с внешней памятью данных разрешается установкой бита SRE регистра MCUCR. По сравнению с обращением к внутренней памяти, обращение к внешней требует дополнительно одного цикла на каждый байт. Это означает, что для выполнения команд LD, ST, LDS, STS, PUSH и POP требуется дополнительно машинный цикл. Если стек размещен во внешней памяти данных, то прерывания, вызов процедур и возвраты требуют по два дополнительных цикла, поскольку в стеке перемещается содержимое двухбайтного счетчика команд. Если при обмене с внешней памятью данных используется состояние ожидания, то на каждый байт необходимо еще два цикла. Поэтому командам пересылки данных необходимы два дополнительных цикла, тогда как при обработке прерываний, вызове процедур и возвратах требуется на четыре цикла больше, чем указано в описании системы команд.

Практическое освоение системы команд и программирования на языке Ассемблер осуществляется в работах №1-3. Программирование ввода-вывода и подсистемы прерываний изучается в работах №4 – 8.

2. ИНТЕГРИРОВАННАЯ СИСТЕМА ПРОГРАММИРОВАНИЯ AVR Studio V4.10

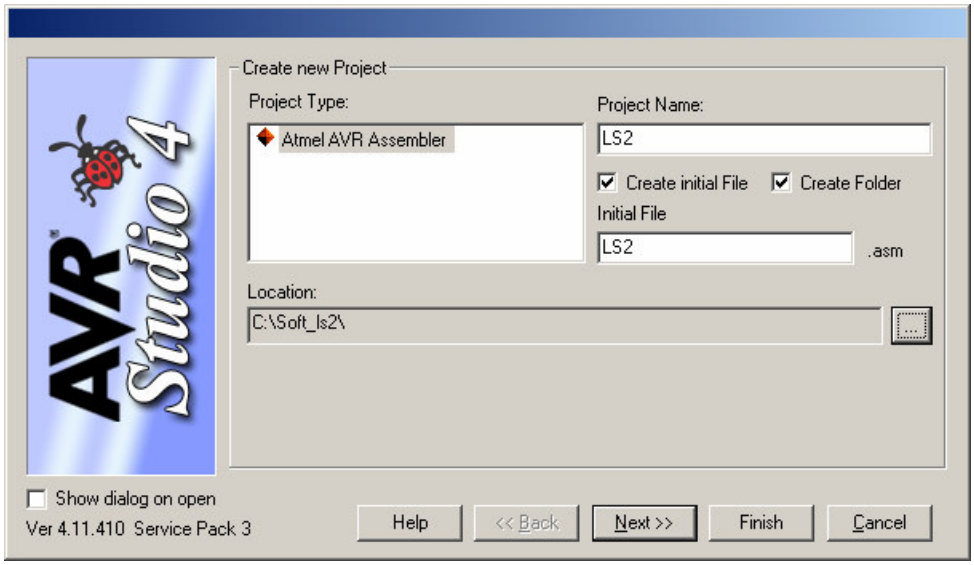
Интегрированная система программирования включает редактор текста, макроассемблер, редактор связей и символический отладчик. Система позволяет разрабатывать целевую программу и отлаживать ее в реальном масштабе времени с использованием аппаратных ресурсов платы контроллера. После создания рабочей программы разработчик имеет возможность, загрузить программу в память микроконтроллера. Отладка программ производится в исходном тексте, причем на каждом шаге можно наблюдать за изменениями внутренних ресурсов микроконтроллера и модифицировать их.

При программировании в среде AVR Studio надо выполнить стандартную последовательность действий:

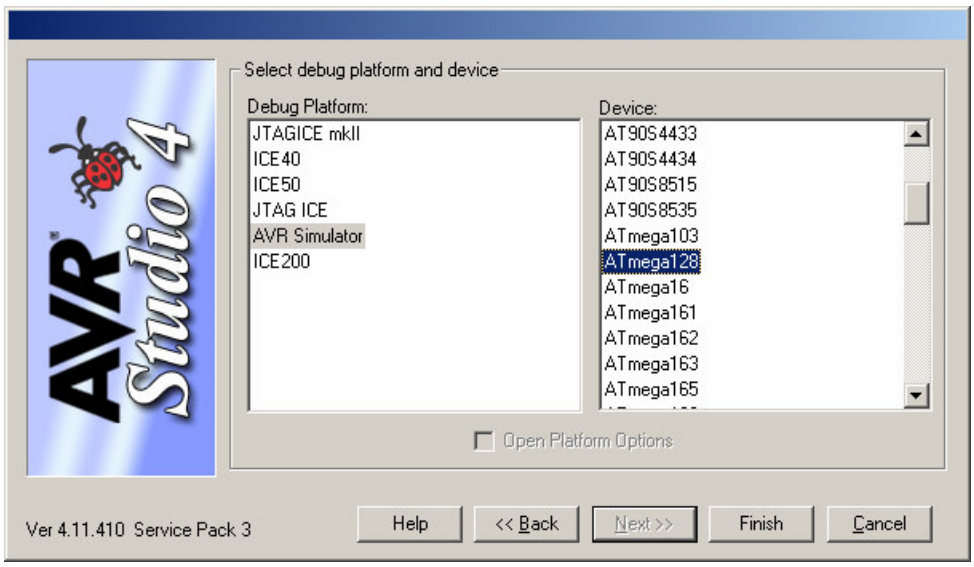
- создание проекта
- создание файла программы
- компиляция
- симуляция
- загрузка hex-кода в микроконтроллер

2.1. СОЗДАНИЕ ПРОЕКТА.

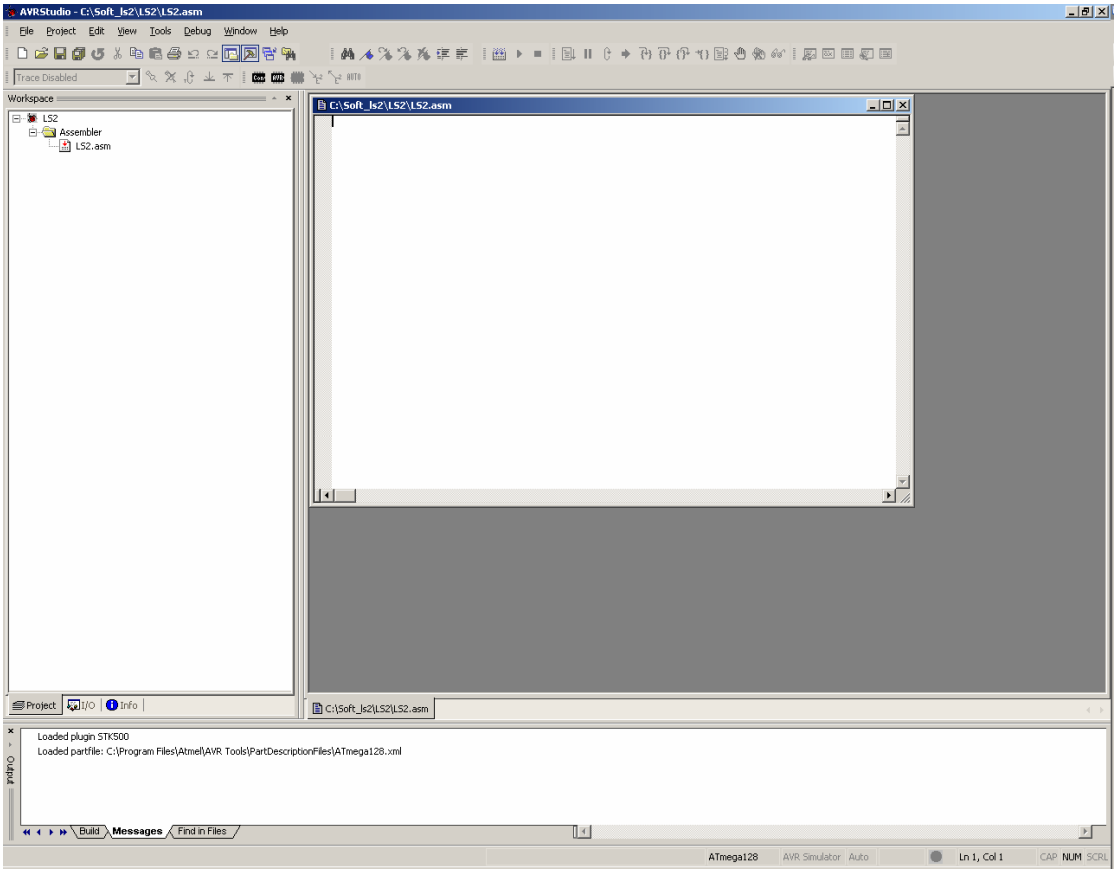
Создание проекта начинается с выбора строки меню Project/New Project. В открывшемся окне «Create new Project» надо указать имя проекта, (в нашем случае – LS2), имя файла программы ls2.asm появится автоматически. В строке Location выбираем папку для сохранения проектов.



После нажатия кнопки «Next» открывается окно «Select debug platform and device», где выбирается отладочная платформа (симулятор или эмулятор) и тип микроконтроллера.



Выбираем в качестве отладочной платформы AVR Simulator и микроконтроллер ATmega128. После нажатия кнопки “Finish” переходим в рабочее окно программы.



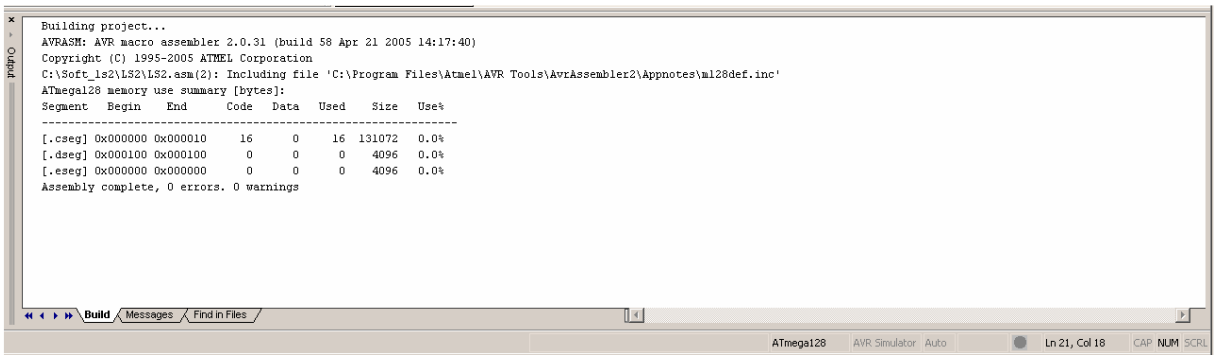
В правом окне вводим исходный текст программы.

```
.device Atmega128
.include    “m128def.inc”

;вывод светодиода подключен к линии PE7

;инициализация стека
      ldi   r16,High(RAMEND)
      out   SPH,r16           ;устанавливаем указатель стека
      ldi   r16,Low(RAMEND)   ;на конечный адрес ОЗУ контроллера
      out   SPL,r16


;настройка линии светодиода на выход
      ldi   r16,(1<<PE7)      ;устанавливаем бит PE7 порта DDRE в 1
      out   DDRE,r16



;Основная программа
main:
      cbi   PORTE,PE7         ;записав в бит PE7 порта PORTE 0 включаем
                                    ;светодиод
      rjmp  main
```

Этот пример включает светодиод в лабораторном стенде ЛС-2.

2.2. КОМПИЛЯЦИЯ ПРОЕКТА.

Компиляция проекта производится командой Project/Build. Процесс компиляции отображается в окне Output в нижней части экрана.



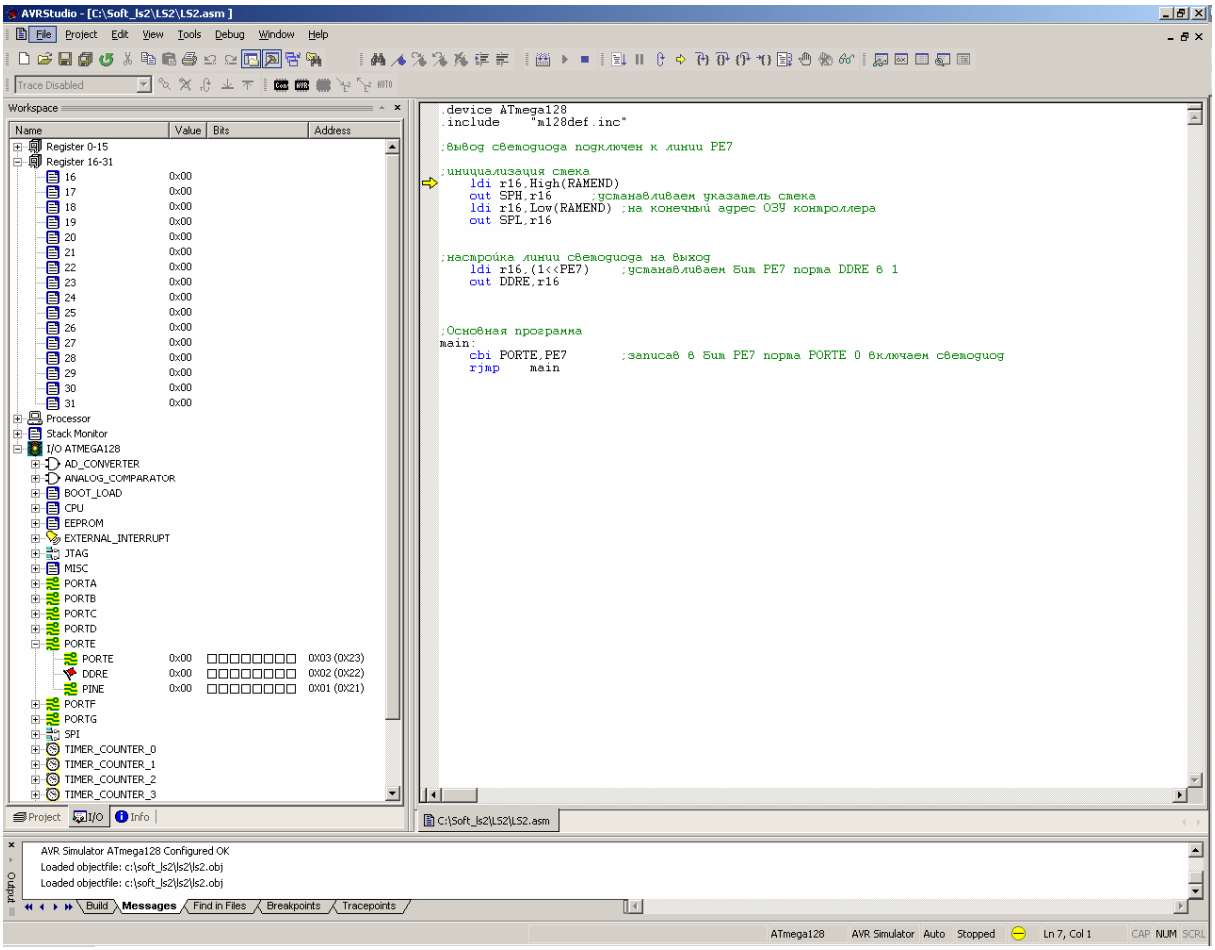
Если программа набрана без ошибок, компилятор создаст файл для прошивки в микроконтроллер ls2.hex и файл для симулятора.

В противном случае будут показаны строки, в которых обнаружена ошибка. После их исправления процесс компиляции повторяем заново.

Мы получили файл для микроконтроллера, который можно запрограммировать и убедиться в работоспособности написанной программы. Но для учебных целей лучше запустить симулятор и провести пошаговую отладку программы.

2.3.СИМУЛЯЦИЯ ПРОЕКТА.

Переход в режим симуляции производится командой Debug/Start Debugging.

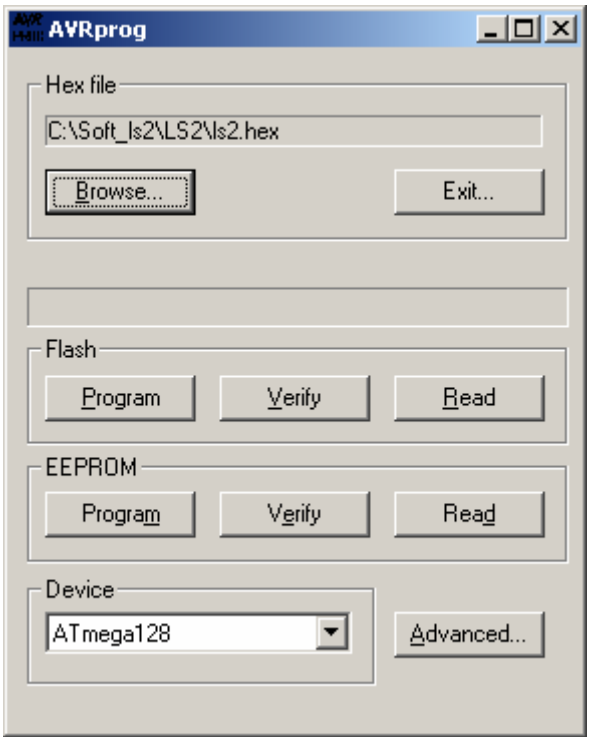


В левой части окна появляются внутренние модули микропроцессора (порты, счетчики, регистры и др.) и их значения, которые можно изменять.

В правом окне появляется желтая стрелка, указывающая на следующую исполняемую команду. По команде Debug/Step Intro(F11) выполняется команда, на которую указывает стрелка. Выполняя пошагово команды можно наблюдать в левом окне изменение содержимого задействованных регистров и портов и находить ошибки в программе.

2.4.ЗАПИСЬ ПРОГРАММЫ В МИКРОКОНТРОЛЛЕР.

Запись программы в микроконтроллер производится программой AVRProg, вызываемой командой Tools/AVRProg. До вызова программы подготовьте лабораторный стенд ЛС-2 к работе в соответствии с паспортом.



После вызова программы AVR Prog кнопкой Browse находим и открываем файл ls2.hex. Кнопкой Program секции Flash загружаем файл ls2.hex во Flash-память микроконтроллера. После успешной записи микроконтроллер начинает выполнять записанную программу.

После успешного выполнения программы на лабораторном стенде ЛС-2 проект можно считать завершенным.

3. ОПИСАНИЕ ЛАБОРАТОРНОГО СТЕНДА ЛС-2.

Лабораторный стенд предназначен для практического изучения микроконтроллеров ATmega128 фирмы Atmel.

Для питания платы требуется нестабилизированный источник напряжения +9..+15В / 300 мА, возможно использование стандартного сетевого адаптера.

На плате установлен микроконтроллер ATmega128, имеющий возможность внутрисхемной загрузки *Flash* памяти программ. Загрузка внутренней памяти программ (*Flash*) и энергонезависимой памяти данных (*EEPROM*) производится через разъем XP2. Для загрузки используется кабель программирования AVRProg. Кабель подключается к персональному компьютеру через COM-порт

В процессе загрузки линии портов микроконтроллера, используемые для этой цели, отключаются от разъемов. В качестве коммутатора использована микросхема CD4066.

Аппаратные ресурсы платы:

- микроконтроллер ATmega128 (DD1) с кварцевым резонатором 6 МГц и резонатором 32768 Гц для реализации часов реального времени;
- стабилизатор напряжения питания (DA1);
- разъем для подключения кабеля программирования (XP2);
- разъем для подключения интерфейса RS232 (XP12), схема драйвера – преобразователя уровней сигналов (DD3);
- разъем для подключения интерфейса SPI (XP8);
- коммутатор (DD2), отключающий сигнальные линии микроконтроллера TxD, RxD, TCK, используемые для загрузки *Flash*, от внешних устройств в процессе программирования;
- разъем для подключения входного аналогового сигнала (XP9), поступающего на вход АЦП микроконтроллера;
- разъем для подключения выходного аналогового сигнала (XP10), поступающего с входа ШИМ микроконтроллера;
- потенциометр для формирования статического аналогового сигнала на входе АЦП и неинвертирующем входе аналогового компаратора;
- цепь формирования опорного напряжения, с возможностью установки интегрального источника опорного напряжения (DA2);
- клавиатура 3x4 (KB0-KB6), столбцы подключены к линии +5В через резисторы;
- светодиод (VD4), управляемый сигналом от микроконтроллера; подключен к линии +5В через резистор, для зажигания нужно подать сигнал логического нуля;
- трехразрядный семисегментный дисплей с общим анодом;
- звуковой пьезодинамик (ZQ3).

Соответствие ресурсов ввода/вывода и портов ATmega приведено в табл.3.1.

Таблица 3.1. Функции портов микроконтроллера

Имя порта	Номер контакта	Имя сигнала	Описание
PB0	10	SS#	Slave Select для интерфейса SPI
PB1	11	TCK	Тактовый сигнал SPI
PB2	12	MOSI	Линия данных интерфейса SPI
PB3	13	MISO	Линия данных интерфейса SPI
PB6	16	OC1B	Выход таймера 1 при работе в режимах Output Compare и PWM
PC0 – PC7	35 – 42	D0 – D7	Шина сегментов для индикатора, активный низкий уровень
PD0 – PD3	25 – 28	KB0 – KB3	Шины строк клавиатуры
PD4 – PD6	29 – 31	KB4 – KB6	Шины столбцов клавиатуры
PD7	32	Beep	Выход управления пьезодинамиком
PE0	2	TDI	Вход RxD интерфейса RS232
PE1	3	TDO	Выход TxD интерфейса RS232
PE4 – PE6	6 – 8	DIG0 – DIG2	Сигналы управления разрядами индикатора, активный низкий уровень
PE7	9	LED	Сигнал управления светодиодом, активный низкий уровень
PF0	61	AN_VAR	Аналоговый статический сигнал, сформированный потенциометром
PF1	62	AN_EXT	Аналоговый вход, сигнал поступает от разъема XP3 через буфер DA4.A

Краткое описание управления ресурсами платы.

Интерфейсный обмен стандарта RS232 обеспечивается по линиям RxD, TxD микроконтроллера. Согласование уровней сигналов обеспечивается драйвером MAX232A (DD3). Соединительный кабель подключается к разъему типа DB-9F.

В процессе загрузки Flash линии RxD (PE0) и TxD (PE1), используемые для передачи данных, отключаются от внешних цепей.

Для связи по интерфейсу SPI предусмотрен разъем XP8.

В процессе загрузки Flash линия SCK (PB1), используемая для стробирования, отключается от разъема XP8.

На плате используются 2 входа АЦП. На вход PF0 подается напряжение с потенциометра R29, в диапазоне 0..5В. На вход PF1 подается внешний аналоговый сигнал от разъема XP9 через буферный операционный усилитель DA4.A.

В качестве источника опорного напряжения используется напряжение 2,56В с интегрального источника опорного напряжения в корпусе TO92 (DA3).

Для формирования выходного аналогового сигнала используется интегрирующая R-C цепь, подключенная к выходу таймера OC1B. Выходное напряжение изменяется пропорционально скважности импульсов на выходе OC1B. Для формирования импульсов рекомендуется использовать режимы *Output Compare* и *PWM*.

Клавиатура с организацией 3x4 подключена к порту PD. Столбцы (PD4 – PD6) подключены к шине питания через резисторы. Сканирование клавиатуры рекомендуется

проводить поочередной подачей сигнала логического нуля на строки (PD0 – PD3), считывая на каждом шаге состояние столбцов.

Светодиодный семисегментный дисплей подключен по схеме с мультиплексированным управлением. В каждый момент времени может быть включен один разряд индикатора. Выбор разряда индикатора производится подачей сигнала логического нуля на выход порта PE4 – PE6 (DIG0 – DIG2), который открывает транзисторный ключ (VT1 – VT3). Управляющий код символа устанавливается на порту C. Включение сегмента на индикаторе соответствует уровню логического нуля на линии порта C. Соответствие сигналов порта C и сегментов индикатора приведено в табл.3.2.

Таблица 3.2. Управление семисегментным дисплеем

Вывод порта B	PB0	PB1	PB2	PB3	PB4	PB5	PB6	PB7
Дисплей	a	b	C	d	e	f	g	точка

РАБОТА №1

МИКРОКОНТРОЛЛЕР ATmega128: методы адресации, команды передачи данных и управления

Цель работы: знакомство с интегрированной средой программирования; изучение методов адресации, команд передачи данных и управления.

Введение

При создании программы для микроконтроллера на языке Ассемблер разработчик оперирует программно доступными ресурсами микропроцессорной системы. У микроконтроллера ATmega128 эти ресурсы включают: программно доступные регистры микроконтроллера, внутреннюю память данных, внешнюю память данных. Перечисленные ресурсы изображены на рис. В2, В3.

Каждая команда языка Ассемблер сообщает процессору выполняемую операцию и методы доступа к операндам. Командная строка Ассемблера включает метку (символический адрес), мнемонику (символическое имя) команды, поле операндов, комментарий. Имя команды однозначно связано с выполняемой ею операцией.

Методы адресации представляют собой набор механизмов доступа к операндам. Одни из них просты и поэтому приводят к компактному формату команды и быстрому доступу к операнду, но объем доступных с их помощью ресурсов ограничен. Другие методы адресации позволяют оперировать со всеми имеющимися в системе ресурсами, но команда получается длинной, на ее ввод и выполнение тратится много времени. Набор методов адресации в каждой системе команд является компромиссным сочетанием известных механизмов адресации, выбранных проектировщиками архитектуры исходя из набора решаемых задач.

Заметим, что при двух операндах и приемнике результата имеет место трехадресная команда и каждый адрес формируется с использованием собственного метода адресации. Если адрес операнда или приемника результата не указан в команде, а подразумевается, то имеет место неявная адресация.

На рис.1.1 – 1.12 приведены схемы методов адресации микроконтроллера ATmega128.

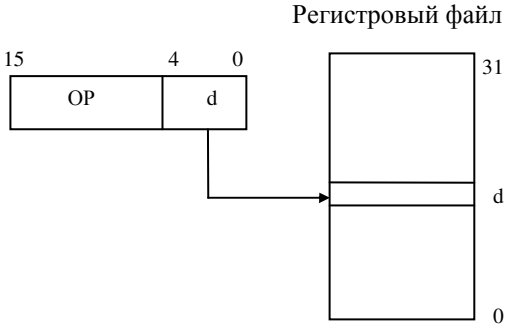


Рис.1.1. Регистровая адресация (один регистр общего назначения)

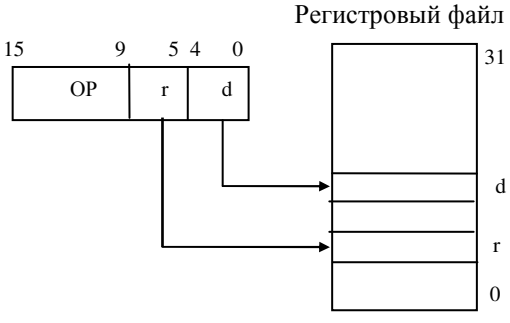


Рис.1.2. Регистровая адресация (два регистра общего назначения)

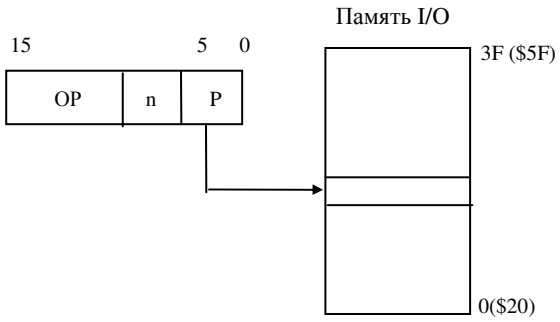


Рис.1.3. Регистровая адресация (регистры ввода-вывода)

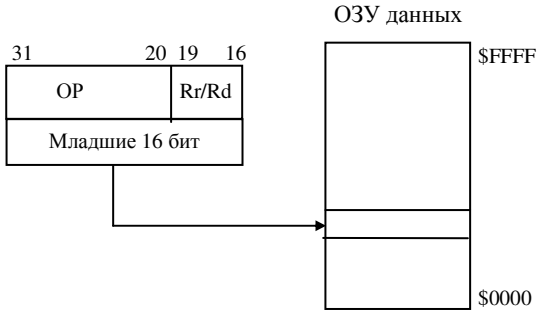


Рис.1.4. Прямая адресация данных

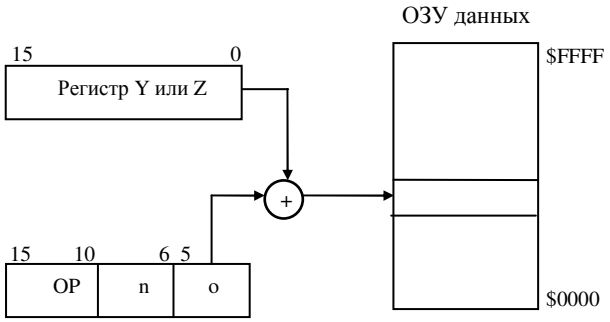


Рис.1.5. Косвенная адресация данных со смещением

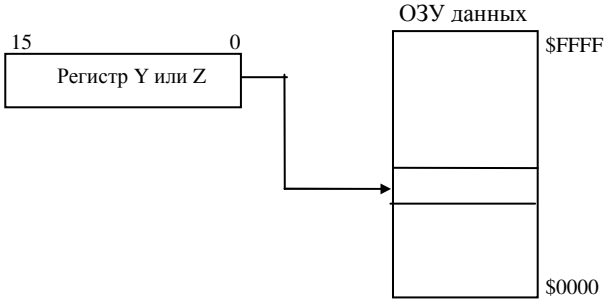


Рис.1.6. Косвенная адресация данных

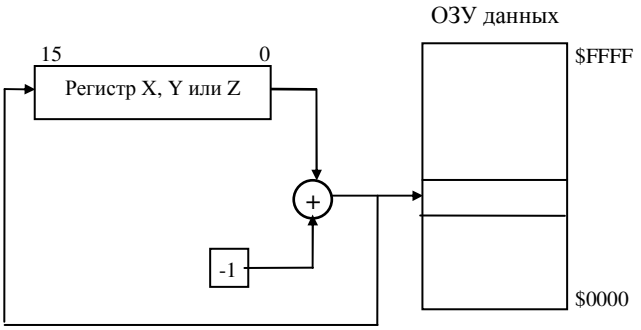


Рис.1.7. Косвенная адресация данных с преддекрементом

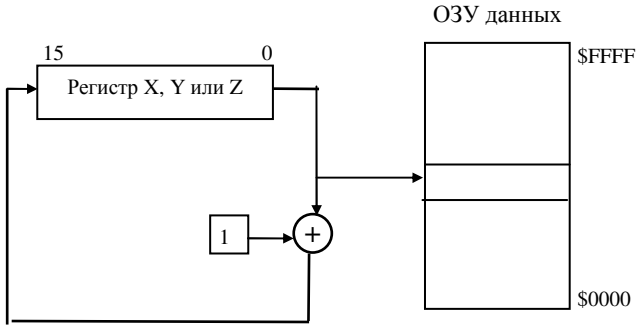


Рис.1.8. Косвенная адресация данных с постинкрементом

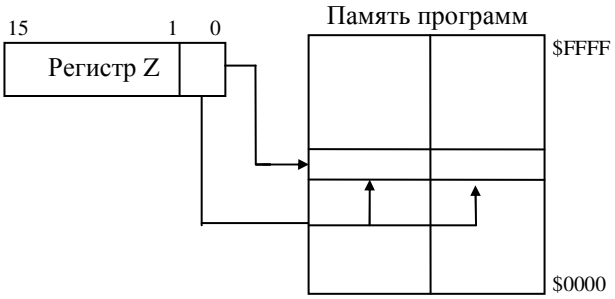


Рис.1.9. Адресация константы в памяти программ в командах LPM и ELPM

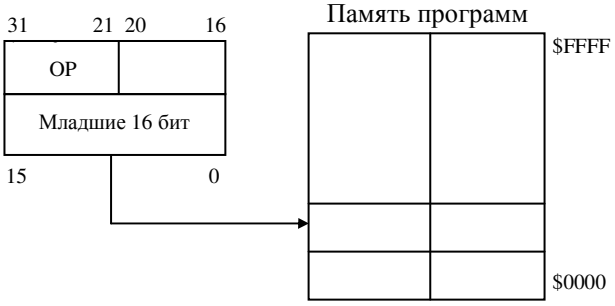


Рис.1.10. Прямая адресация памяти программ в командах JMP и CALL

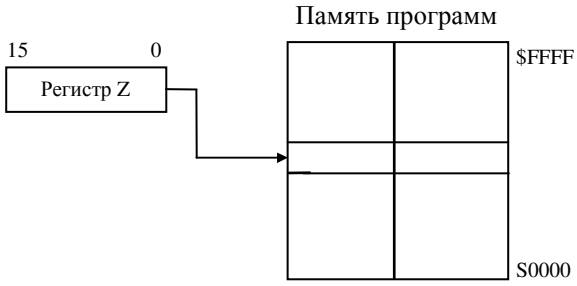


Рис.1.11. Косвенная адресация памяти программ в командах IJMP и ICALL

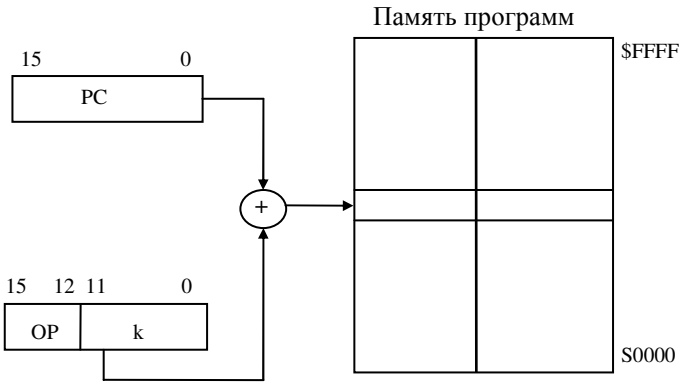


Рис.1.12. Относительная адресация памяти программ в командах RJMP и RCALL

Команды передачи данных приведены в Табл.1.1. Из таблицы видно, что набор этих команд представляет собой сочетание восьми операций с различными методами адресации.

Таблица 1.1. Команды пересылки данных

Мнемо-ника	**Операн-ды**	**Описание**	**Операция**	**Флаги**	**Кол-во циклов**
ELPM		Расширенная загрузка из памяти программ в регистр R0	R0 ← (Z+RAMPZ)	Нет	3
MOV	Rd,Rr 0≤d≤31 0≤r≤31	Копировать регистр	Rd ← Rr	Нет	1
LDI	Rd,K 16≤d≤31 0≤k≤255	Загрузить непосредственное значение	Rd ← K	Нет	1
LDS	Rd,k 0≤d≤31 0≤k≤65535	Загрузить из ОЗУ	Rd ← (k)	Нет	3
LD	Rd,X 0≤d≤31	Загрузить косвенно	Rd ← (X)	Нет	2
LD	Rd,X+ 0≤d≤31	Загрузить косвенно с постинкрементом	Rd ← (X), X ← X+1	Нет	2
LD	Rd,-X 0≤d≤31	Загрузить косвенно с преддекрементом	X ← X - 1, Rd ← (X)	Нет	2
LD	Rd,Y 0≤d≤31	Загрузить косвенно	Rd ← (Y),	Нет	2
LD	Rd,Y+ 0≤d≤31	Загрузить косвенно с постинкрементом	Rd ← (Y), Y ← Y+1	Нет	2
LD	Rd,-Y 0≤d≤31	Загрузить косвенно с преддекрементом	Y ← Y - 1, Rd ← (Y)	Нет	2

LDD	Rd,Y+q 0≤d≤31 0≤q≤63	Загрузить косвенно со смещением	Rd ← (Y+q)	Нет	2
LD	Rd,Z 0≤d≤31	Загрузить косвенно	Rd ← (Z)	Нет	2
LD	Rd,Z+ 0≤d≤31	Загрузить косвенно с постинкрементом	Rd ← (Z), Z ← Z+1	Нет	2
LD	Rd,-Z 0≤d≤31	Загрузить косвенно с преддекрементом	Z ← Z - 1, Rd ← (Z)	Нет	2
LDD	Rd,Z+q 0≤d≤31 0≤q≤31	Загрузить косвенно со смещением	Rd ← (Z+q)	Нет	2
STS	k,Rr 0≤r≤31 0≤k≤65535	Загрузить непосредственно в ОЗУ	(k) ← Rr	Нет	3
ST	X,Rr 0≤r≤31	Записать косвенно	(X) ← Rr	Нет	2
ST	X+,Rr 0≤r≤31	Записать косвенно с постинкрементом	(X) ← Rr, X ← X + 1	Нет	2
ST	-X,Rr 0≤r≤31	Записать косвенно с преддекрементом	X ← X - 1, (X) ← Rr	Нет	2
ST	Y,Rr 0≤r≤31	Записать косвенно	(Y) ← Rr	Нет	2
ST	Y+,Rr 0≤r≤31	Записать косвенно с постинкрементом	(Y) ← Rr, Y ← Y + 1	Нет	2
ST	-Y,Rr 0≤r≤31	Записать косвенно с преддекрементом	Y ← Y - 1, (Y) ← Rr	Нет	2
STD	Y+q,Rr 0≤r≤31 0≤q≤63	Записать косвенно со смещением	(Y+q) ← Rr	Нет	2
ST	Z,Rr 0≤r≤31	Записать косвенно	(Z) ← Rr	Нет	2
ST	Z+,Rr 0≤r≤31	Записать косвенно с постинкрементом	(Z) ← Rr, Z ← Z + 1	Нет	2
ST	-Z,Rr 0≤r≤31	Записать косвенно с преддекрементом	Z ← Z - 1, (Z) ← Rr	Нет	2
STD	Z+q,Rr 0≤r≤31 0≤q≤63	Записать косвенно со смещением	(Z+q) ← Rr	Нет	2
LPM		Загрузить байт из памяти программ	R0 ← (Z)	Нет	3
IN	Rd,P 0≤d≤31 0≤P≤63	Загрузить данные из порта I/O в регистр	Rd ← P	Нет	1
OUT	P,Rr 0≤r≤31 0≤P≤63	Записать данные из регистра в порт I/O	P ← Rr	Нет	1
PUSH	Rr 0≤r≤31	Сохранить регистр в стеке	STACK ← Rr	Нет	2
POP	Rd 0≤d≤31	Выгрузить регистр из стека	Rd ← STACK	Нет	2

Команды ELPM и LPM выполняют загрузку памяти программ. Команда MOV копирует содержимое одного регистра общего назначения в другой. Команда LDI загружает регистр общего назначения непосредственно константой (только R16 – R32), а команды LD выполняют загрузку регистра из ячейки ОЗУ при косвенной адресации, используя в качестве источника адреса регистры X, Y, Z, а также варианты с постинкрементом и преддекрементом источника. Команда LDD выполняет эту же операцию при помощи косвенной адресации со смещением. Команда LDS загружает в регистр общего назначения содержимое прямо адресуемой ячейки памяти данных.

Команда ST выполняют обратную операцию относительно LD – заносит в косвенно адресуемую ячейку памяти данных содержимое регистра общего назначения, используя также варианты с постинкрементом и преддекрементом адреса. Команда STD выполняет эту же операцию при помощи косвенной адресации со смещением, а команда STS прямо адресует ячейку ОЗУ.

Команда IN заносит содержимое регистра ввода-вывода в регистр общего назначения, а команда OUT выполняет обратную операцию.

Команда PUSH сохраняет содержимое регистра в стеке, а команда POP выполняет обратную операцию. При выполнении этих команд кроме программного счетчика изменяется и значение указателя стека SP.

Обратите внимание, что метод адресации не указывается

Команды передачи управления (переходов) приведены в Табл.1.2.

Таблица 1.2. Команды переходов

Мнемо-ника	**Операн-ды**	**Описание**	**Операция**	**Флаги**	**Кол-во циклов**
RJMP	k -2K≤k≤2K	Перейти относительно	PC ← PC + k + 1	Нет	2
LJMP		Перейти косвенно	PC ← Z	Нет	2
JMP	k 0≤k≤4M	Перейти	PC ← k	Нет	3
RCALL	K -2K≤k≤2K	Вызвать подпрограмму относительно	PC ← PC + k + 1	Нет	3
ICALL		Вызвать подпрограмму косвенно	PC ← Z	Нет	3
CALL	K 0≤k≤64K	Выполнить длинный вызов подпрограммы	PC ← k	Нет	4
RET		Вернуться из подпрограммы	PC ← STACK	Нет	4
RETI		Вернуться из прерывания	PC ← STACK	1	4
CPSE	Rd,Rr 0≤d≤31, 0≤r≤31	Сравнить и пропустить, если равно	if Rd=Rr then PC ← PC + 2 (or 3)	Нет	1/2/3
SBRC	Rr,b 0≤r≤31 0≤b≤7	Пропустить, если бит в регистре очищен	if Rr(b)=0 then PC ← PC + 2 (or 3)	Нет	1/2/3
SBRS	Rr,b 0≤r≤31 0≤b≤7	Пропустить, если бит в регистре установлен	if Rr(b)=1 then PC ← PC + 2 (or 3)	Нет	1/2/3
SBIC	P,b 0≤P≤31 0≤b≤7	Пропустить, если бит в регистре I/O очищен	if I/OP(b)=0 then PC ← PC + 2 (or 3)	Нет	1/2/3
SBIS	P,b 0≤r≤31 0≤b≤7	Пропустить, если бит в регистре I/O установлен	if I/OP(b)=1 then PC ← PC + 2 (or 3)	Нет	1/2/3
BRBS	s,k 0≤s≤7 -64≤k≤+63	Перейти, если бит в регистре статуса установлен	if SREG(s)=1 then PC ← PC + k + 1	Нет	1/2
BRBC	s,k 0≤s≤7 -64≤k≤+63	Перейти, если бит в регистре статуса очищен	if SREG(s)=0 then PC ← PC + k + 1	Нет	1/2
BREQ	k -64≤k≤+63	Перейти, если равно	if Rd=Rr(Z=1) then PC ← PC + k + 1	Нет	1/2
BRNE	k -64≤k≤+63	Перейти, если не равно	if Rd≠Rr(Z=0) then PC ← PC + k + 1	Нет	1/2
BRCS	k -64≤k≤+63	Перейти, если флаг переноса установлен	if C=1 then PC ← PC + k + 1	Нет	1/2
BRCC	k -64≤k≤+63	Перейти, если флаг переноса очищен	if C=0 then PC ← PC + k + 1	Нет	1/2
BRSH	k -64≤k≤+63	Перейти, если равно или больше (без знака)	if Rd≥Rr (C=0) then PC ← PC + k + 1	Нет	1/2

BRLO	k -64≤k≤+63	Перейти, если меньше (без знака)	if Rd<Rr (C=1) then PC ← PC + k + 1	Нет	1/2
BRMI	k -64≤k≤+63	Перейти, если минус	if N=1 then PC ← PC + k + 1	Нет	1/2
BRPL	k -64≤k≤+63	Перейти, если плюс	if N=0 then PC ← PC + k + 1	Нет	1/2
BRGE	k -64≤k≤+63	Перейти, если больше или равно (с учетом знака)	if Rd≥Rr (N⊕V=0) then PC ← PC + k + 1	Нет	1/2
BRLT	k -64≤k≤+63	Перейти, если меньше чем (со знаком)	if Rd<Rr (N⊕V=1) then PC ← PC + k + 1	Нет	1/2
BRHS	k -64≤k≤+63	Перейти, если флаг полупереноса установлен	if H=1 then PC ← PC + k + 1	Нет	1/2
BRHC	k -64≤k≤+63	Перейти, если флаг полупереноса очищен	if H=0 then PC ← PC + k + 1	Нет	1/2
BRTS	k -64≤k≤+63	Перейти, если флаг T установлен	if T=1 then PC ← PC + k + 1	Нет	1/2
BRTC	k -64≤k≤+63	Перейти, если флаг T очищен	if T=0 then PC ← PC + k + 1	Нет	1/2
BRVS	k -64≤k≤+63	Перейти, если флаг переполнения установлен	if V=1 then PC ← PC + k + 1	Нет	1/2
BRVC	k -64≤k≤+63	Перейти, если флаг переполнения очищен	if V=0 then PC ← PC + k + 1	Нет	1/2
BRIE	k -64≤k≤+63	Перейти, если глобальное прерывание разрешено	if I=1 then PC ← PC + k + 1	Нет	1/2
BRID	k -64≤k≤+63	Перейти, если глобальное прерывание запрещено	if I=0 then PC ← PC + k + 1	Нет	1/2

Из табл.1.1 и табл.1.2 видно, что команды пересылки данных и команды переходов значения флагов регистра SREG не изменяют.

ВЫПОЛНЕНИЕ ОСНОВНОГО ЗАДАНИЯ

1. Изучить описание структуры микроконтроллера ATmega128 и интегрированной системы его программирования на языке Ассемблер AVRStudio.

2. Изучить реализуемые микроконтроллером способы адресации и команды пересылки данных.

3. Запустить на персональном компьютере интегрированную систему программирования AVRStudio. Командой Project|New Project создать новый проект Lesson1. С использованием редактора текста создать демонстрационную программу:

```
.device ATmega128
.include "m128def.inc"

;***** Инициализация указателя стека ********************************
          ldi  R20, $FF      ; загрузка промежуточного регистра R20 младшим
                             ; байтом адреса начала стека
          out  SPL, R20      ; загрузка младшего байта указателя стека
          ldi  R20, $10      ; загрузка промежуточного регистра R20 старшим
                             ; байтом адреса начала стека
          out  SPH, R20      ; загрузка старшего байта указатель стека
;***** Выполнение команд пересылок **********************************

          ldi  R20, $57      ; загрузка регистра R20 константой

          ldi  R30, $00      ; загрузка регистровой пары Z (R30,R31) адресом $0100
          ldi  R31, $01      ; по которому расположена первая ячейка внутреннего ОЗУ

          st   Z, R20        ; загрузка косвенно адресуемой ячейки ОЗУ с адресом $0100
                             ; значением из регистра R20 ($57)

          lds  R19, $0100    ; загрузка регистра R19 из ячейки ОЗУ с адресом $0100
          sts  $0101, R19    ; загрузка ячейки с адресом $0101 из регистра R19
```

```
          call Rout          ; вызов подпрограммы Rout
loop:
          rjmp loop          ; зацикливание программы

;***** Подпрограмма копирования значения из памяти программ ***************
Rout:
          push R30           ; сохранение указателя Z в стеке
          push R31           ;

          ldi  R20, $00      ; загрузка регистра R20 новой константой

          ldi  R30, $80      ; загрузка регистровой пары Z удвоенным адресом ячейки
          ldi  R31, $00      ; памяти программ

          lpm                ; загрузка регистра R0 значением ячейки памяти программ
                             ; с адресом $0040

          cpse R0, R20       ; пропустить следующую команду если значения регистров
                             ; R0 и R20 равно

          sts  $0102, R0     ; загрузка ячейки ОЗУ с адресом $0102 из регистра R0

          pop R31            ; извлечение указателя Z из стека
          pop R30            ;

          ret                ; возврат из подпрограммы
;*****************************************************************************
```

4. Выполнить команду Project/Build для компиляции проекта.

5. С помощью команды Debug/Start Debugging запустить симулятор. Командой View/Memory открыть окно с ячейками памяти программ. Прямым редактированием занести любой код в ячейку $40 памяти программ.

6. Выполнить программу по шагам, выполняя команду Debug/Step Intro(F11) . После выполнения текущей команды курсор в окне редактора текста указывает на следующую команду.

7. Проверить правильность пересылки данных.

8. Составить программу выполнения заданной преподавателем последовательности операций передачи данных, провести пошаговый прогон программы, продемонстрировать полученный результат.

РАБОТА №2

МИКРОКОНТРОЛЛЕР ATmega128: команды обработки данных

Цель работы: изучение команд арифметических и логических операций, сдвигов и битовых операций, практическое освоение приемов программирования на ассемблере, изучение директив.

Введение

Для обработки данных микроконтроллер ATmega128 использует группу команд, реализующих арифметические и логические операции, сдвиги и операции над отдельными битами. Арифметические операции являются операциями над 8-разрядными целыми числами. Для выполнения целочисленных операций над длинными словами служат команды сложения и вычитания с учетом флага переноса

Арифметические и логические команды приведены в табл.2.1. и табл.2.2.

Таблица 2.1. Арифметические и логические команды

Мнемо-ника	Операнды	Описание	Операция	Флаги	К-во циклов
ADD	Rd,Rr 0≤d≤31 0≤r≤31	Сложить без переноса	Rd ← Rd + Rr	Z, C, N, V, H	1
ADC	Rd,Rr 0≤d≤31 0≤r≤31	Сложить с переносом	Rd ← Rd + Rr + C	Z, C, N, V, H	1
ADIW	Rd,K d∣∈{24,26,28,30} 0≤K≤63	Сложить непосредственное значение со словом	Rdh:Rdl ← Rdh:Rdl + K	Z, C, N, V	2
SUB	Rd,Rr 0≤d≤31 0≤r≤31	Вычесть без заема	Rd ← Rd - Rr	Z, C, N, V, H	1
SUBI	Rd, K 16≤d≤31 0≤K≤255	Вычесть непосредственное значение	Rd ← Rd - K	Z, C, N, V, H	1
SBC	Rd, Rr 0≤d≤31 0≤r≤31	Вычесть с заемом	Rd ← Rd - Rr - C	Z, C, N, V, H	1
SBCI	Rd, K 16≤d≤31 0≤K≤255	Вычесть непосредственное значение с заемом	Rd ← Rd - K - C	Z, C, N, V, H	1
SBIW	Rd, K d∣∈{24,26,28,30} 0≤K≤63	Вычесть непосредственное значение из слова	Rdh:Rdl ← Rdh:Rdl - K	Z, C, N, V	2
AND	Rd, Rr 0≤d≤31 0≤r≤31	Выполнить логическое AND	Rd ← Rd • Rr	Z, N, V	1
ANDI	Rd, K 16≤d≤31 0≤K≤255	Выполнить логическое AND	Rd ← Rd • K	Z, N, V	1
OR	Rd, Rr 0≤d≤31 0≤r≤31	Выполнить логическое OR	Rd ← Rd ∨ Rr	Z, N, V	1
ORI	Rd, K 16≤d≤31 0≤K≤255	Выполнить логическое OR с непосредственным значением	Rd ← Rd ∨ K	Z, N, V	1

EOR	Rd, Rr 0≤d≤31 0≤r≤31	Выполнить исключающее OR	Rd ← Rd ⊕ Rr	Z, N, V	1
COM	Rd 0≤d≤31	Выполнить дополнение до единицы	Rd ← SFF - Rd	Z, C, N, V	1
NEG	Rd 0≤d≤31	Выполнить дополнение до двух	Rd ← S00 - Rd	Z, C, N, V, H	1
SBR	Rd, K 16≤d≤31 0≤K≤255	Установить биты в регистре	Rd ← Rd ∨ K	Z, N, V	1
CBR	Rd, K 16≤d≤31 0≤K≤255	Очистить биты в регистре	Rd ← Rd • (SFF - K)	Z, N, V	1
INC	Rd 0≤d≤31	Инкрементировать	Rd ← Rd + 1	Z, N, V	1
DEC	Rd 0≤d≤31	Декрементировать	Rd ← Rd - 1	Z, N, V	1
TST	Rd 0≤d≤31	Проверить на ноль или минус	Rd ← Rd • Rd	Z, N, V	1
CLR	Rd 0≤d≤31	Очистить регистр	Rd ← Rd ⊕ Rd	Z, N, V	1
SER	Rd 16≤d≤31	Установить все биты регистра	Rd ← SFF	нет	1
CP	Rd, Rr 0≤d≤31 0≤r≤31	Сравнить	Rd - Rr	Z, C, N, V, H	1
CPC	Rd, Rr 0≤d≤31 0≤r≤31	Сравнить с учетом переноса	Rd - Rr - C	Z, C, N, V, H	1
CPI	Rd, K 16≤d≤31 0≤K≤255	Сравнить с константой	Rd - K	Z, C, N, V, H	1

Таблица 2.2. Команды сдвигов и операций с битами

Мнемо-ника	**Операн-ды**	**Описание**	**Операция**	**Флаги**	**Кол-во циклов**
LSL	Rd 0≤d≤31	Логически сдвинуть влево	Rd(n+1) ← Rd(n), Rd(0) ←0, C ← Rd(7)	Z,C,N,V,H	1
LSR	Rd 0≤d≤31	Логически сдвинуть вправо	Rd(n) ← Rd(n+1), Rd(7) ←0, C ← Rd(0)	Z,C,N,V	1
ROL	Rd 0≤d≤31	Сдвинуть влево через перенос	Rd(0) ←C, Rd(n+1) ← Rd(n), C ← Rd(7)	Z,C,N,V,H	1
ROR	Rd 0≤d≤31	Сдвинуть вправо через перенос	Rd(7) ←C, Rd(n) ← Rd(n+1), C ← Rd(0)	Z,C,N,V	1
ASR	Rd 0≤d≤31	Арифметически сдвинуть вправо	Rd(n) ← Rd(n+1), C ← Rd(0), n=0...6	Z,C,N,V	1
SWAP	Rd 0≤d≤31	Поменять нибблы местами	Rd(3...0) ↔ Rd(7...4)	Нет	1
BSET	s 0≤s≤7	Установить флаг	SREG(s)← 1	SREG(s)	1
BCLR	s 0≤s≤7	Очистить флаг	SREG(s)← 0	SREG(s)	1

SBI	P,b 0≤P≤31 0≤b≤7	Установить бит в регистр I/O	I/O(P,b)← 1	Нет	2
CBI	P,b 0≤P≤31 0≤b≤7	Очистить бит в регистре I/O	I/O(P,b)← 0	Нет	2
BST	Rd,b 0≤d≤31 0≤b≤7	Переписать бит из регистра во флаг T	T← Rd(b)	T	1
BLD	Rd,b 0≤d≤31 0≤b≤7	Загрузить T флаг в бит регистра	Rd(b) ← T	Нет	1
SEC		Установить флаг переноса	C← 1	C	1
CLC		Очистить флаг переноса	C← 0	C	1
SEN		Установить флаг отрицательного значения	N← 1	N	1
CLN		Очистить флаг отрицательного значения	N← 0	N	1
SEZ		Установить флаг нулевого значения	Z← 1	Z	1
CLZ		Очистить флаг нулевого значения	Z← 0	Z	1
SEI		Установить флаг глобального прерывания	I← 1	I	1
CLI		Очистить флаг глобального прерывания	I← 0	I	1
SES		Установить флаг знака	S← 1	S	1
CLS		Очистить флаг знака	S← 0	S	1
SEV		Установить флаг переполнения	V← 1	V	1
CLV		Очистить флаг переполнения	V← 0	V	1
SET		Установить флаг T	T← 1	T	1
CLT		Очистить флаг T	T← 0	T	1
SEH		Установить флаг полупереноса	H← 1	H	1
CLH		Очистить флаг полупереноса	H← 0	H	1
NOP		Выполнить холостую команду		Нет	1
SLEEP		Установить режим SLEEP	См. описание команды	Нет	1
WDR		Сбросить сторожевой таймер	См. описание команды	Нет	1

ВЫПОЛНЕНИЕ РАБОТЫ

1. Изучить набор команд арифметических, логических, битовых операций и сдвигов, выполняемых микроконтроллером ATmega128.

2. Запустить на персональном компьютере интегрированную систему программирования AVRStudio. Командой Project|New Project создать новый проект Lesson2. С использованием редактора текста создать демонстрационную программу:

```
.device ATmega128
.include "m128def.inc"

;***** Определение констант  ********************************
.EQU      IniX    =$0100
.EQU      IniY    =$0110
.EQU      IniZ    =$0120
.EQU      SPadr   =$10FF
.EQU      NCircle =16
```

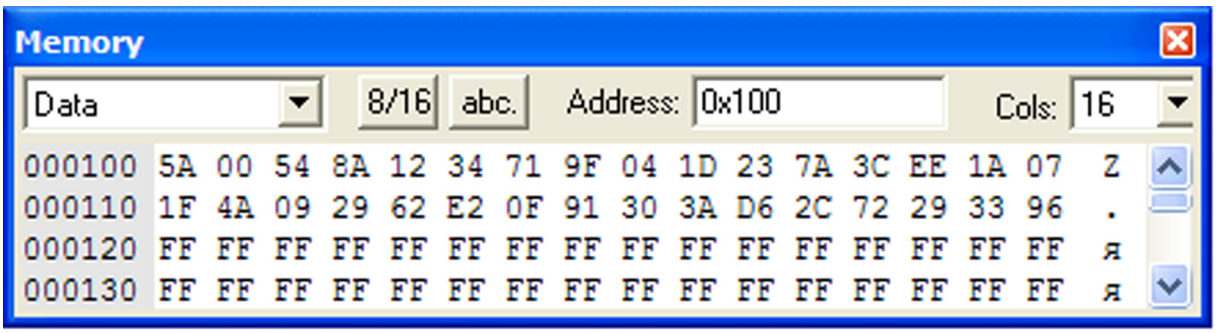
```
;***** Инициализация указателя стека *********************************
          ldi  R20, $FF       ; загрузка промежуточного регистра R20 младшим
                              ; байтом адреса начала стека
          out  SPL, R20       ; загрузка младшего байта указателя стека
          ldi  R20, $10       ; загрузка промежуточного регистра R20 старшим
                              ; байтом адреса начала стека
          out  SPH, R20       ; загрузка старшего байта указатель стека

;***** Инициализация указателей, счетчика циклов и флага управления переносом
          ldi  R26,low(IniX)  ; загрузка рег. X значением указателя на память данных
          ldi  R27,high(IniX) ;
          ldi  R28,low(IniY)  ; загрузка рег. Y значением указателя на память данных
          ldi  R29,high(IniY) ;
          ldi  R30,low(IniZ)  ; загрузка рег. Z значением указателя на память данных
          ldi  R31,high(IniZ) ;
          ldi  R16,Ncircle    ; в R16 – счетчик циклов=2*N_ЧИСЕЛ
          ldi  R17,$00        ; в R17 – 0
          ldi  R18,$01        ; в R18 – 1 в младший бит

;***** Сложение массивов 16-разрядных чисел ************************
LP1:
          sbrs R17, 0
          clc
LP2:      ld   R3, X+         ; загрузка байта 1-операнда
          ld   R4, Y+         ; загрузка байта 2-операнда
          adc  R3, R4         ; сложение байтов операндов с учетом переноса
          st   Z+, R3         ; запись результата в память
          eor  R17, R18       ; чередование 0 и 1 в младшем бите R17
          dec  R16            ; декремент счетчика байтов
          brne LP1            ; зацикливание, если счетчик байтов не равен нулю

loop:
          rjmp loop           ; зацикливание программы для исключения зависаний
```

Эта программа читает восемь пар двухбайтных операндов из ОЗУ данных, складывает операнды побайтно с учетом флага переноса и посылает результаты также в ОЗУ. В программе использованы директивы EQU, с помощью которых определяются имена констант, встречающихся далее в тексте. Этими константами являются значения адреса областей ОЗУ, где хранятся операнды и результаты, количество циклов сложения. Область первых операндов начинается с адреса $0100, вторых операндов - $0110, результата - $0120. Количество циклов равно 2байта*8операндов=16.

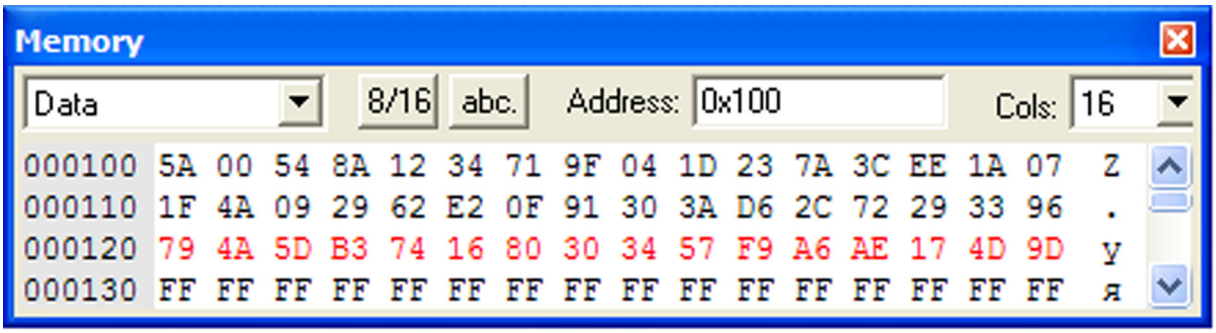
Для сокращения числа команд внутри цикла предусмотрена одинаковая обработка первых байтов операндов, которые складываются без учета переноса, и вторых байтов, складываемых с учетом значения бита C. Единый цикл использует команду ADC, но дополнительно привлекается бит 0 регистра R17, значение которого изменяется при каждом проходе. При R17.0==0 (обработка первых байтов) бит переноса обнуляется. Такой алгоритм применим только для 16-разрядных операндов. Для работы программы необходим счетчик, три указателя на области ОЗУ и флаг обнуления бита переноса. В качестве счетчика циклов используется регистр R16. Его содержимое декрементируется и проверяется командой BRNE, которая организует цикл. В качестве указателей на ячейки-источники операндов используются регистры X и Y. На текущую ячейку-приемник результата указывает регистр Z.

3. Выполнить команду Project/Build для компиляции проекта.

4. При успешной компиляции с помощью команды Debug/Start Debugging запустить симулятор. Командой View/Memory открыть окно с ячейками ОЗУ. Используя прямое редактирование ОЗУ данных (матрица Data в окне Memory) занести в ячейки с адреса $0100 шестнадцать байтов любого кода, которые будут представлять восемь первых двухбайтных операндов. С адреса $0110 занести шестнадцать байтов другого кода, которые будут представлять восемь вторых двухбайтных операндов. Например такие:



5. Выполнить программу по шагам, выполняя команду Debug/Step Intro(F11). Наблюдать изменение значений в регистрах микроконтроллера и ячейках ОЗУ данных. При указанных выше операндах результаты должны быть следующими:



6. Составить программу выполнения задания, провести пошаговое выполнение программы, продемонстрировать полученный результат.

Задания

1. Выполнить попарно сложение 16-разрядных чисел со знаком, содержащихся в двух массивах объемом по 15 чисел с начальными адресами $xx, $yy, результаты сохранить в массиве с начальным адресом $zz. Номера чисел, при сложении которых возникло переполнение, записать в регистры R20 и R21. Общее количество переполнении указать в регистре R22.

2. Выполнить попарное вычитание 16-разрядных чисел со знаком, содержащихся в двух массивах объемом по 10 чисел с начальными адресами $xx, $yy, результаты сохранить в массиве с начальным адресом $zz. При переполнении результат округлять до максимального положительного или отрицательного значения. Общее количество округлений указать в регистре R22.

3. Выбрать числа с минимальным и максимальным значением из массива, содержащего 40 16-разрядных чисел со знаком, которые размещены в памяти, начиная с адреса $xx. Минимальное и максимальное число записать в регистры R20, R21, а порядковые номера этих чисел в исходном массиве данных - в регистры R22 и R23.

4. Выполнить сравнение 8-разрядных чисел, содержащихся в массиве объемом из 50 чисел с начальным адресом $xx, с эталоном, хранящимся регистре R17. Числа, несовпадающие с эталоном, разместить в массиве с начальным адресом $yy. Общее число несовпадающих чисел записать в регистр R20.

РАБОТА №3

ЦИФРОВЫЕ СИСТЕМЫ НА БАЗЕ МИКРОКОНТРОЛЛЕРА ATmega128: реализация и обслуживание подсистемы прерываний

Цель работы: изучение организации прерываний в микроконтроллере ATmega128 и обслуживания подсистемы прерываний.

Введение

Микроконтроллеры ATmega128 используют 34 источника прерывания. Эти прерывания и механизм сброса располагают отдельными векторами в пространстве памяти программ. Каждому прерыванию присвоен свой бит разрешения, который должен быть установлен совместно с битом I регистра статуса SREG.

Младшие адреса пространства памяти программ автоматически определяются как векторы сброса и прерываний.

Полный перечень векторов представлен в Таблице 3.1. Перечень отражает также уровень приоритета для каждого прерывания. Прерывания с младшими адресами имеют больший уровень приоритета: RESET имеет наивысший уровень приоритета, следующим является запрос внешнего прерывания INT0 –и т.д.

Таблица 3.1. Векторы сброса и прерываний

№ п.п.	Адрес	Источник	Наименование
1	$0000	RESET	Сброс по сигналу Reset, включению питания и сторожевому таймеру
2	$0002	INT0	Запрос внешнего прерывания 0
3	$0004	INT1	Запрос внешнего прерывания 1
4	$0006	INT2	Запрос внешнего прерывания 2
5	$0008	INT3	Запрос внешнего прерывания 3
6	$000A	INT4	Запрос внешнего прерывания 4
7	$000C	INT5	Запрос внешнего прерывания 5
8	$0006	INT6	Запрос внешнего прерывания 6
9	$0010	INT7	Запрос внешнего прерывания 7
10	$0012	TIMER2 COMP	Совпадение таймера/счетчика T2
11	$0014	TIMER2 OVF	Переполнение таймера/счетчика T2
12	$0016	TIMER1 CAPT	Захват таймера/счетчика T1
13	$0018	TIMER1 COMPA	Совпадение «A» таймера/счетчика T1
14	$001A	TIMER1 COMPB	Совпадение «B» таймера/счетчика T1
15	$001C	TIMER1 OVF	Переполнение таймера/счетчика T1
16	$001E	TIMER0 COMP	Совпадение при сравнении таймера/счетчика T0
17	$0020	TIMER0 OVF	Переполнение таймера/счетчика T0
18	$0022	SPI, STC	Завершена пересылка порта SPI
19	$0024	UART0, RX	Завершение приема UART0
20	$0026	UART0, UDRE	Регистр данных UART0 пуст
21	$0028	UART0, TX	Завершение передачи UART0
22	$002A	ADC	Завершение AD преобразования
23	$002C	EE READY	Готовность EEPROM
24	$002E	ANALOG COMP	Срабатывание аналогового компаратора
25	$0030	TIMER1 COMPC	Совпадение «C» таймера/счетчика T1
26	$0032	TIMER3 CAPT	Переполнение таймера/счетчика T3
27	$0034	TIMER3 COMPA	Совпадение «A» таймера/счетчика T3
28	$0036	TIMER3 COMPB	Совпадение «B» таймера/счетчика T3
29	$0038	TIMER3 COMPC	Совпадение «C» таймера/счетчика T3
30	$003A	TIMER3 OVF	Переполнение таймера/счетчика T3
31	$003C	USART1, RX	Завершение приема UART
32	$003E	USART1, UDRE	Регистр данных UART1 пуст
33	$0040	USART1, TX	Завершение передачи UART1
34	$0042	TWI	Прерывание от модуля TWI
35	$0044	SPM_RDY	Готовность SPM

Микроконтроллеры ATmega128 содержат два специальных 8-разрядных регистра масок прерываний: регистр масок внешних прерываний EIMSK (External Interrupt Mask) и регистр масок

прерываний таймеров/счетчиков TIMSK (Timer/Counter Interrupt Mask). Кроме того, в регистрах управления блоками ввода-вывода могут существовать и другие биты разрешения и биты масок.

При возникновении прерывания бит I разрешения всех прерываний (в регистре SREG) очищается и все прерывания запрещаются. Процедура прерывания может установить бит I, чтобы разрешить вложенные прерывания. Выход из процедуры обработки прерывания происходит по команде RETI, которая устанавливает бит I (=1).

Когда в счетчик команд загружен вектор процедуры обработки прерывания, соответствующий флаг, вызвавший прерывание, аппаратно очищается. Некоторые флаги прерываний можно очистить программно.

Если условия прерывания возникли, когда соответствующий бит разрешения очищен, флаг этого прерывания будет установлен и сохранен в таком состоянии, пока прерывание не будет разрешено, или флаг не будет очищен программно.

Если условия прерываний возникли, когда очищен бит разрешения всех прерываний, флаги прерываний будут установлены и сохранены в таком состоянии, пока все прерывания не будут разрешены и обработаны в порядке приоритетов.

Прерывания по уровню сигнала, флага не имеют и условия прерывания имеют место, пока активен внешний сигнал.

Регистр масок внешних прерываний – EIMSK

	7	6	5	4	3	2	1	0	
$39 ($59)	**INT7**	**INT6**	**INT5**	**INT4**	**INT3**	**INT2**	**INT1**	**INT0**	**EIMSK**
Исходное значение	0	0	0	0	0	0	0	0	

Bits 7..0 – Разрешение внешних прерываний INT7..INT0. При установленных битах INT7-INT0 и установленном бите I регистра статуса (SREG) разрешаются прерывания по соответствующим выводам сигналов внешних прерываний. Активность сигнала по любому из этих выводов вызовет запрос прерывания, даже если вывод назначен как выход. Это обеспечивает возможность организации программного прерывания.

Регистр флагов внешних прерываний - EIFR

	7	6	5	4	3	2	1	0	
$38 ($58)	**INTF7**	**INTF6**	**INTF5**	**INTF4**	**INTF3**	**INTF2**	**INTF1**	**INTF0**	**EIFR**
Исходное значение	0	0	0	0	0	0	0	0	

Bits 7..0 – Флаги внешних прерываний INTF7 – INTF0. В случае поступления запроса на прерывание на какой-либо из выводов INT7 – INT0 устанавливается (=1) соответствующий флаг прерывания INTF7 – INTF0. Если бит I регистра SREG и соответствующий бит разрешения (INT7 – INT0) регистра EIMSK установлены, то выполняется переход по вектору прерывания. При возврате из процедуры прерывания флаг очищается. Кроме того, флаг можно очистить, записав в него логическую 1.Обратите внимание, что программно установить флаг внешнего прерывания, записав в него «1», невозможно!

Регистр управления внешними прерываниями – EICRA и EICRB

	7	6	5	4	3	2	1	0	
$6A	**ISC31**	**ISC30**	**ISC21**	**ISC20**	**ISC11**	**ISC10**	**ISC01**	**ISC00**	**EICRA**
Исходное значение	0	0	0	0	0	0	0	0	

	7	6	5	4	3	2	1	0	
$3A ($5A)	**ISC71**	**ISC70**	**ISC61**	**ISC60**	**ISC51**	**ISC50**	**ISC41**	**ISC40**	**EICRB**
Исходное значение	0	0	0	0	0	0	0	0	

Bits7..0 - ISCX1, ISCX0: Биты управления опознаванием внешних прерываний INT7-INT0. Запросы внешних прерываний на выводах INT7 – INT0 являются активными, если установлен флаг I регистра SREG и установлена соответствующая маска в регистре EIMSK. Запрос прерывания по логическому уровню или фронтам определяется в соответствии со следующей таблицей:

Таблица 4.2. Управление опознаванием прерывания

ISCx1	ISCx0	Описание
0	0	Запрос прерывания генерируется низким уровнем на INTx
0	1	Зарезервировано
1	0	Запрос прерывания генерируется спадающим фронтом на INTx
1	1	Запрос прерывания генерируется нарастающим фронтом на INTx

Примечание: X может быть равен 7, 6, 5,4,3,2,1 или 0

При изменении битов ISCx1/ISCx0 прерывание должно быть запрещено очисткой бита разрешения в регистре GIMSK. В ином случае может произойти прерывание в момент изменения бита.

Входы прерываний INTx периодически опрашиваются на предмет наличия запроса. Если внешний запрос прерывания фиксируется по фронту, то для гарантированного срабатывания длительность импульса должна быть больше, чем один период тактовой частоты процессора. Заметим, что частота процессора может быть меньше частоты XTAL из-за возможного наличия делителя. Запрос прерывания по логическому уровню должен продолжаться, пока выполняется текущая инструкция, и он будет зафиксирован. Запрос прерывания по логическому уровню, если он разрешен, будет генерировать запрос прерывания до тех пор, пока на входе удерживается низкий уровень.

Регистр масок прерывания по таймерам/счетчикам - TIMSK

	7	6	5	4	3	2	1	0	
$37 ($57)	**OCIE2**	**TOIE2**	**TICIE1**	**OCIE1A**	**OCIE1B**	**TOIE1**	**OCIE0**	**TOIE0**	**TIMSK**
Исходное значение	0	0	0	0	0	0	0	0	

Bit 7 - OCIE2: Разрешение прерывания по совпадению Таймера 2. При установленном бите OCIE2 и установленном бите I регистра статуса разрешается прерывание по совпадению содержимого регистра сравнения и состояния Таймера 2. Соответствующее прерывание (с вектором $0012) имеет место, если произойдет совпадение при сравнении содержимого регистра сравнения и состояния таймера/счетчика2. В регистре флагов прерывания TIFR устанавливается флаг OCF2.

Bit 6 - TOIE2: - Разрешение прерывания по переполнению Таймера 2. При установленном бите TOIE2 и установленном бите I регистра статуса разрешается прерывание по переполнению Таймера 2. Соответствующее прерывание (с вектором $0014) выполняется если произойдет переполнение Таймера 2. В регистре флагов TIFR устанавливается флаг TOV2 переполнения Таймера 2.

Bit 5 - TICIE1: - Разрешение прерывания по захвату Таймера 1. При установленном бите TICIE1 и установленном бите I регистра статуса разрешается прерывание по захвату Таймера 1. Соответствующее прерывание (с вектором $0016) выполняется, если произошел захват по выводу 29, PD4(IC1). В регистре флагов TIFR устанавливается флаг ICF1 захвата Таймера 1.

Bit 4 - OCIE1A: - Разрешение прерывания по совпадению регистра A с Таймером 1. При установленном бите OCIE1A и установленном бите I регистра статуса разрешается прерывание по совпадению регистра A с состоянием Таймера 1. Соответствующее прерывание (с вектором $0018) выполняется, если имеется совпадение содержимого регистра A с состоянием Таймера 1. В регистре флагов TIFR устанавливается флаг OCF1A совпадения регистра A с Таймером 1.

Bit 3 - OCIE1B: - Разрешение прерывания по совпадению регистра B с Таймером 1. При установленном бите OCIE1B и установленном бите I регистра статуса разрешается прерывание по совпадению регистра B с состоянием Таймера 1. Соответствующее прерывание (с вектором $001A) выполняется, если имеется совпадение содержимого регистра B с состоянием Таймера 1. В регистре флагов TIFR устанавливается флаг OCF1B совпадения регистра B с Таймером 1.

Bit 2 - TOIE1: - Разрешение прерывания по переполнению Таймера 1. При установленном бите TOIE1 и установленном бите I регистра статуса разрешается прерывание по переполнению Таймера 1. Соответствующее прерывание (с вектором $001C) выполняется, если произойдет переполнение Таймера 1. В регистре флагов TIFR устанавливается флаг TOV1 переполнения Таймера 1.

Bit 1 - OCIE0: - Разрешение прерывания по совпадению Таймера 0. При установленном бите OCIE0 и установленном бите I регистра статуса разрешается прерывание по совпадению содержимого регистра сравнения и состояния Таймера 0. Соответствующее прерывание (с вектором $001E) выполняется, если произойдет совпадение при сравнении содержимого регистра сравнения и состояния Таймера 0. В регистре флагов прерывания TIFR устанавливается флаг OCF0 совпадения Таймера 0.

Bit 0 - TOIE0: - Разрешение прерывания по переполнению Таймера 0. При установленном бите TOIE0 и установленном бите I регистра статуса разрешается прерывание по переполнению Таймера 0. Соответствующее прерывание (с вектором $0020) выполняется, если произойдет переполнение Таймера 0. В регистре флагов TIFR устанавливается флаг TOV0 переполнения Таймера 0.

Регистр масок прерывания по таймерам/счетчикам - ETIMSK

	7	6	5	4	3	2	1	0	
$7D	**-**	**-**	**TICIE3**	**OCIE3A**	**OCIE3B**	**TOIE3**	**OCIE3C**	**OCIE1C**	**ETIMSK**
Исходное значение	0	0	0	0	0	0	0	0	

Bit 7, Bit 6 - Зарезервировано.

Bit 5 – TICIE3: - Разрешение прерывания по захвату Таймера 3. При установленном бите TICIE3 и установленном бите I регистра статуса разрешается прерывание по захвату Таймера 3. Соответствующее прерывание (с вектором $0032) выполняется, если произошел захват по выводу 9, PE7(ICP3). В регистре флагов ETIFR устанавливается флаг ICF3C захвата Таймера 3.

Bit 4 – OCIE3A: - Разрешение прерывания по совпадению регистра A с Таймером 3. При установленном бите OCIE3A и установленном бите I регистра статуса разрешается прерывание по совпадению регистра A с состоянием Таймера 3. Соответствующее прерывание (с вектором $0034) выполняется, если имеется совпадение содержимого регистра A с состоянием Таймера 3. В регистре флагов ETIFR устанавливается флаг OCF3A совпадения регистра A с Таймером 3.

Bit 3 – OCIE3B: - Разрешение прерывания по совпадению регистра B с Таймером 3. При установленном бите OCIE3B и установленном бите I регистра статуса разрешается прерывание по совпадению регистра B с состоянием Таймера 3. Соответствующее прерывание (с вектором $0036) выполняется, если имеется совпадение содержимого регистра B с состоянием Таймера 3. В регистре флагов ETIFR устанавливается флаг OCF3B совпадения регистра B с Таймером 3.

Bit 2 – TOIE3: - Разрешение прерывания по переполнению Таймера 3. При установленном бите TOIE3 и установленном бите I регистра статуса разрешается прерывание по переполнению Таймера 3. Соответствующее прерывание (с вектором $003A) выполняется, если произойдет переполнение Таймера 3. В регистре флагов ETIFR устанавливается флаг TOV3 переполнения Таймера 3.

Bit 1 – OCIE3C: - Разрешение прерывания по совпадению регистра C с Таймером 3. При установленном бите OCIE3C и установленном бите I регистра статуса разрешается прерывание по совпадению содержимого регистра сравнения C с состоянием Таймера 3. Соответствующее прерывание (с вектором $0038) выполняется, если произойдет совпадение при сравнении содержимого регистра сравнения C и состояния Таймера 3. В регистре флагов прерывания ETIFR устанавливается флаг OCF3C совпадения Таймера 3.

Bit 0 – OCIE1C: - Разрешение прерывания по совпадению регистра C с Таймером 1. При установленном бите OCIE1C и установленном бите I регистра статуса разрешается прерывание по совпадению содержимого регистра сравнения C с состоянием Таймера 1. Соответствующее прерывание (с вектором $0030) выполняется, если произойдет совпадение при сравнении содержимого регистра сравнения C и состояния Таймера 1. В регистре флагов прерывания ETIFR устанавливается флаг OCF1C совпадения Таймера 1.

Регистр флагов прерываний по таймерам/счетчикам –TIFR

	7	6	5	4	3	2	1	0	
$36 ($56)	**OCF2**	**TOV2**	**ICF1**	**OCF1A**	**OCF1B**	**TOV1**	**OCF0**	**TOV0**	**TIFR**
Исходное значение	0	0	0	0	0	0	0	0	

Bit 7 - OCF2: - Флаг 2 совпадения Таймера 2 и данных OCR2. Бит OCF2 устанавливается при совпадении состояния Таймера 2 и содержимого регистра OCR2. Бит OCF2 аппаратно очищается при обработке соответствующего вектора прерывания. Возможна очистка бита записью во флаг логической 1. При установленных битах I в регистре SREG, OCIE2 и OCF2 выполняется прерывание по совпадению Таймера 2.

Bit 6 - TOV2: - Флаг переполнения Таймера 2. Бит TOV2 устанавливается при переполнении Таймера 2. Он аппаратно очищается при обработке соответствующего вектора прерывания. Возможна очистка бита записью во флаг логической 1. При установленном бите I в регистре SREG, установленных битах OCIE2 и TOV2 выполняется прерывание по переполнению Таймера 2. В режиме PWM этот бит устанавливается, при переходе через $00.

Bit 5 - ICF1: - Флаг захвата входа. Бит ICF1 устанавливается в случае захвата входа и показывает, что значение Таймера 1 переслано во входной регистр захвата ICR1. Бит очищается аппаратно при обработке соответствующего вектора прерывания. Возможна очистка бита записью во флаг логической 1.

Bit 4 - OCF1A: - Флаг 1A совпадения выхода. Бит OCF1A устанавливается при совпадении состояния Таймера 1 и содержимого регистра OCR1A. Бит OCF1A аппаратно очищается при обработке соответствующего вектора прерывания. Возможна очистка бита записью во флаг логической 1. При установленном бите I в регистре SREG, установленных битах OCIE1A и OCF1A выполняется прерывание по совпадению выхода Таймера счетчика1.

Bit 3 - OCF1B: - Флаг 1B совпадения выхода. Бит OCF1B устанавливается при совпадении состояния Таймера 1 и содержимого регистра OCR1B. Бит OCF1B аппаратно очищается при обработке соответствующего вектора прерывания. Возможна очистка бита записью во флаг логической 1. При установленном бите I в регистре SREG, установленных битах OCIE1B и OCF1B выполняется прерывание по совпадению выхода Таймера 1.

Bit 2 - TOV1: - Флаг переполнения Таймера 1. Бит TOV1 устанавливается при переполнении Таймера 1. Он аппаратно очищается при обработке соответствующего вектора прерывания. Возможна очистка бита записью во флаг логической 1. При установленном бите I в регистре SREG, установленных битах TOIE1 и TOV1 выполняется прерывание по переполнению Таймера 1.

Bit 1 - OCF0: - Флаг 0 совпадения выхода. Бит OCF0 устанавливается при совпадении состояния Таймера 0 и содержимого регистра OCR0 . Бит OCF0 аппаратно очищается при обработке соответствующего вектора прерывания. Возможна очистка бита записью во флаг логической 1. При установленном бите I в регистре SREG, установленных битах OCIE0 и OCF0 выполняется прерывание по совпадению выхода Таймера 1.

Bit 0 - TOV0: - Флаг переполнения Таймера 0. Бит TOV0 устанавливается при переполнении Таймера 0. Он аппаратно очищается при обработке соответствующего вектора прерывания. Возможна очистка бита записью во флаг логической 1. При установленном бите I в регистре SREG, установленных битах TOIE0 и TOV0 выполняется прерывание по переполнению Таймера 0.

Регистр флагов прерываний по таймерам/счетчикам –ETIFR

	7	6	5	4	3	2	1	0	
$7C	**-**	**-**	**ICF3C**	**OCF3A**	**OCF3B**	**TOV3**	**OCF3C**	**OCF1C**	**ETIFR**
Исходное значение	0	0	0	0	0	0	0	0	

Bit 7, Bit 6 – Зарезервировано.

Bit 5 – ICF3C: - Флаг захвата входа. Бит ICF3C устанавливается в случае захвата входа и показывает, что значение Таймера 3 переслано во входной регистр захвата ICR3. Бит очищается аппаратно при обработке соответствующего вектора прерывания. Возможна очистка бита записью во флаг логической 1.

Bit 4 – OCF3A: - Флаг 3A совпадения выхода. Бит OCF3A устанавливается при совпадении состояния Таймера 3 и содержимого регистра OCR3A. Бит OCF3A аппаратно очищается при обработке соответствующего вектора прерывания. Возможна очистка бита записью во флаг логической 1. При установленном бите I в регистре SREG, установленных битах OCIE3A и OCF3A выполняется прерывание по совпадению выхода Таймера 3.

Bit 3 – OCF3B: - Флаг 3B совпадения выхода. Бит OCF3B устанавливается при совпадении состояния Таймера 3 и содержимого регистра OCR3B. Бит OCF3B аппаратно очищается при обработке соответствующего вектора прерывания. Возможна очистка бита записью во флаг логической 1. При установленном бите I в регистре SREG, установленных битах OCIE3B и OCF3B выполняется прерывание по совпадению выхода Таймера 3.

Bit 2 – TOV3: - Флаг переполнения Таймера 3. Бит TOV3 устанавливается при переполнении Таймера 3. Он аппаратно очищается при обработке соответствующего вектора прерывания. Возможна очистка бита записью во флаг логической 1. При установленном бите I в регистре SREG, установленных битах TOIE3 и TOV3 выполняется прерывание по переполнению Таймера 3.

Bit 1 – OCF3C: - Флаг 3C совпадения выхода. Бит OCF3C устанавливается при совпадении состояния Таймера 3 и содержимого регистра OCR3C. Бит OCF3C аппаратно очищается при обработке соответствующего вектора прерывания. Возможна очистка бита записью во флаг логической 1. При установленном бите I в регистре SREG, установленных битах OCIE3C и OCF3C выполняется прерывание по совпадению выхода Таймера 3.

Bit 0 – OCF1C: - Флаг 1C совпадения выхода. Бит OCF1C устанавливается при совпадении состояния Таймера 1 и содержимого регистра OCR1C. Бит OCF1C аппаратно очищается при обработке соответствующего вектора прерывания. Возможна очистка бита записью во флаг логической 1. При установленном бите I в регистре SREG, установленных битах OCIE1C и OCF1C выполняется прерывание по совпадению выхода Таймера 1.

Обратите внимание, что программно установить флаг таймера-счетчика, записав в него «1», невозможно!

ВЫПОЛНЕНИЕ ОСНОВНОГО ЗАДАНИЯ

1. Изучить функционирование подсистемы прерываний микроконтроллера ATmega128.

2. Запустить на персональном компьютере интегрированную систему программирования AVRStudio. Командой Project|New Project создать новый проект Lesson3. С использованием редактора текста создать демонстрационную программу:

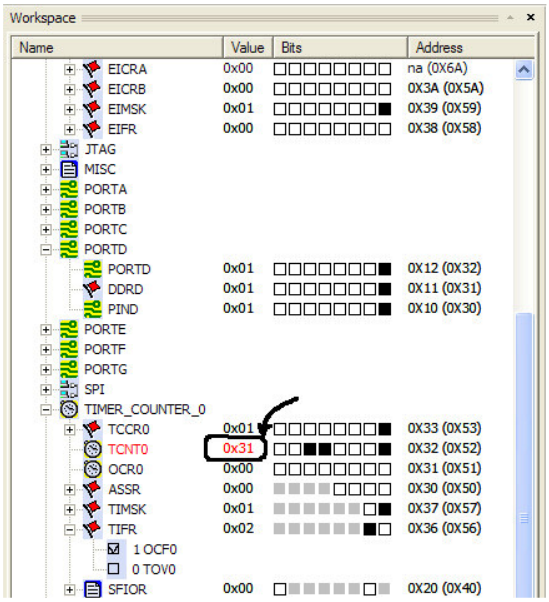
```
.device ATmega128
.include "m128def.inc"
;*****таблица векторов прерываний***************************************
.CSEG
.org     $0000                ;начальный адрес программы
         rjmp       Start     ;переход к рабочей части программы (вектор Reset)
.org     $0002
         rjmp       int_INT0  ;Внешнее прерывание Int0
.org     $0004
         reti                 ;Внешнее прерывание Int1
.org     $0006
         reti                 ;Внешнее прерывание Int2
.org     $0008
         reti                 ;Внешнее прерывание Int3
.org     $000A
         reti                 ;Внешнее прерывание Int4
.org     $000C
         reti                 ;Внешнее прерывание Int5
.org     $000E
         reti                 ;Внешнее прерывание Int6
.org     $0010
         reti                 ;Внешнее прерывание Int7
.org     $0012
         reti                 ;Совпадение таймера/счетчика T2
.org     $0014
         reti                 ;Переполнение таймера/счетчика T2
.org     $0016
```

```
        reti                    ;Захват таймера/счетчика T1
.org    $0018
        reti                    ;Совпадение "A" таймера/счетчика T1
.org    $001A
        reti                    ;Совпадение "B" таймера/счетчика T1
.org    $001C
        reti                    ;Переполнение таймера/счетчика T1
.org    $001E
        reti                    ;Совпадение таймера/счетчика T0
.org    $0020
        rjmp    OvfT0           ;Переполнение таймера/счетчика T0
.org    $0022
        reti                    ;Передача по SPI завершена
.org    $0024
        reti                    ;USART0, прием завершен
.org    $0026
        reti                    ;Регистр данных USART0 пуст
.org    $0028
        reti                    ;USART0, передача завершена
.org    $002A
        reti                    ;Преобразование АЦП завершено
.org    $002C
        reti                    ;Запись в EEPROM завершена
.org    $002E
        reti                    ;Аналоговый компаратор
.org    $0030
        reti                    ;Совпадение "C" таймера/счетчика T1
.org    $0032
        reti                    ;Захват таймера/счетчика T3
.org    $0034
        reti                    ;Совпадение "A" таймера/счетчика T3
.org    $0036
        reti                    ;Совпадение "B" таймера/счетчика T3
.org    $0038
        reti                    ;Совпадение "C" таймера/счетчика T3
.org    $003A
        reti                    ;Переполнение таймера/счетчика T3
.org    $003C
        reti                    ;USART1, прием завершен
.org    $003E
        reti                    ;Регистр данных USART1 пуст
.org    $0040
        reti                    ;USART1, передача завершена
.org    $0042
        reti                    ;Прерывание от модуля TWI
.org    $0044
        reti                    ;Готовность SPM
;***** Основная программа*********************************************
Start:
        ldi     R16,High(RAMEND)    ;инициализация стека
        out     SPH,R16             ;
        ldi     R16,Low(RAMEND)     ;
        out     SPL,R16             ;
;*****настройка линии INT0 для программного симулятора*****************
        in      R16,DDRD            ;
        sbr     R16,(1<<PD0)        ;установка линии порта INT0 на выход
        out     DDRD, R16           ;разрешение работы линии порта INT0 на выход
        sbi     PORTD,PD0           ;установка высокого уровня на выходе INT0
;*****************************************************************
        in      R16,TCCR0           ; установка бита CS00,тактирование таймера
        sbr     R16,(1<<CS00)       ; частотой процессора
        out     TCCR0, R16          ;разрешение работы Таймера 0

        in      R16,TIMSK           ;
        sbr     R16,(1<<TOIE0)      ;установка бита TOIE0
        out     TIMSK, R16  ;разрешение прерывания по переполнению Таймера 0

        in      R16,EIMSK           ;
        sbr     R16,(1<<INT0)       ;установка бита INT0
        out     EIMSK, R16          ;разрешение прерывания INT0 от низкого уровня
                                    ;на входе INT0
```

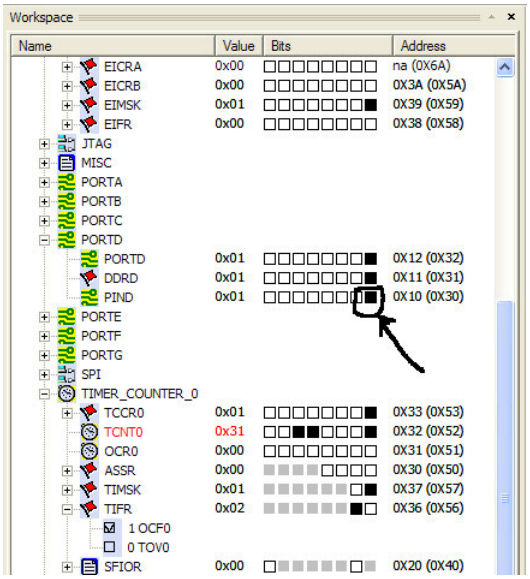
```
        sei                             ;разрешение всех прерываний
loop:
        rjmp  loop                      ;зацикливание программы до прихода прерывания

;***** Обработка прерывания Int0: ***************************************
int_INT0:
        nop
        reti

;***** Обработка прерывания Таймера0 *************************************
OvfT0:
        nop
        reti
```

Программа lesson3 показывает работу процессора по прерываниям. Разрешены прерывания от внешнего вывода INT0 и переполнения Таймера 0. Процедуры обслуживания разрешенных источников не содержат исполняемых команд и подробнее будут рассмотрены в следующих работах.

3. Выполнить компиляцию программы командой Project/Build.
4. При успешной компиляции с помощью команды Debug/Start Debugging запустить симулятор.
5. Выполнить программу по шагам, выполняя команду Debug/Step Intro(F11) до зацикливания loop . В окне Workspace изменяем, значение регистра TCNT0 на FF тем самым вызываем прерывание по переполнению Таймера 0.



6. Выполнить программу по шагам, используя команду Debug/Step Intro(F11) до возврата в цикл loop . В окне Workspace обнуляем, бит 0 регистра PIND тем самым вызываем прерывание INT0.



Задания

1. Составить различные процедуры обслуживания прерываний, выполняемых при переполнении и сравнении Таймеров 1,2,3.

2. Запустить составленные программы на выполнение. Реализовать вызов и выполнение соответствующих процедур обслуживания прерываний, изменяя значения в регистрах Таймеров 1,2,3 и контролируя работу программы с помощью интегрированной системы программирования.

РАБОТА №4

ЦИФРОВЫЕ СИСТЕМЫ НА БАЗЕ МИКРОКОНТРОЛЛЕРА ATmega128: работа с внешними устройствами через параллельные порты ввода вывода – работа с клавиатурой и светодиодным индикатором

Цель работы: получение практических навыков программирования процедур ввода-вывода сигналов внешних устройств (на примере работы с клавиатурой и индикатором).

Введение

Индикатор является неотъемлемой частью любого электронного устройства. Роль индикатора могут выполнять многие устройства от светодиода до TFT панели.

Установленный на плате лабораторного стенда светодиодный индикатор позволяет отображать 3 разряда по 8 сегментов в каждом. Он подключен к портам C и E. Схема подключения показана на рисунке 4.1. Восемь сегментов подключены к выводам PC0-PC7, три разряда к выводам PE4-PE6. Вывод информации на индикатор производится в динамическом режиме, то есть в каждый момент времени включается один разряд из восьми сегментов. Для исключения мерцания индикатора частота включения каждого разряда должна превышать 25Гц. Вывод информации на индикатор производится последовательной установкой “0” на выводах PE4-PE6 и включением нужных сегментов установкой в “0” выводов PC0-PC7. Перед выключением предыдущего и включением следующего разряда выдерживается пауза, от времени которой зависит яркость индикатора.

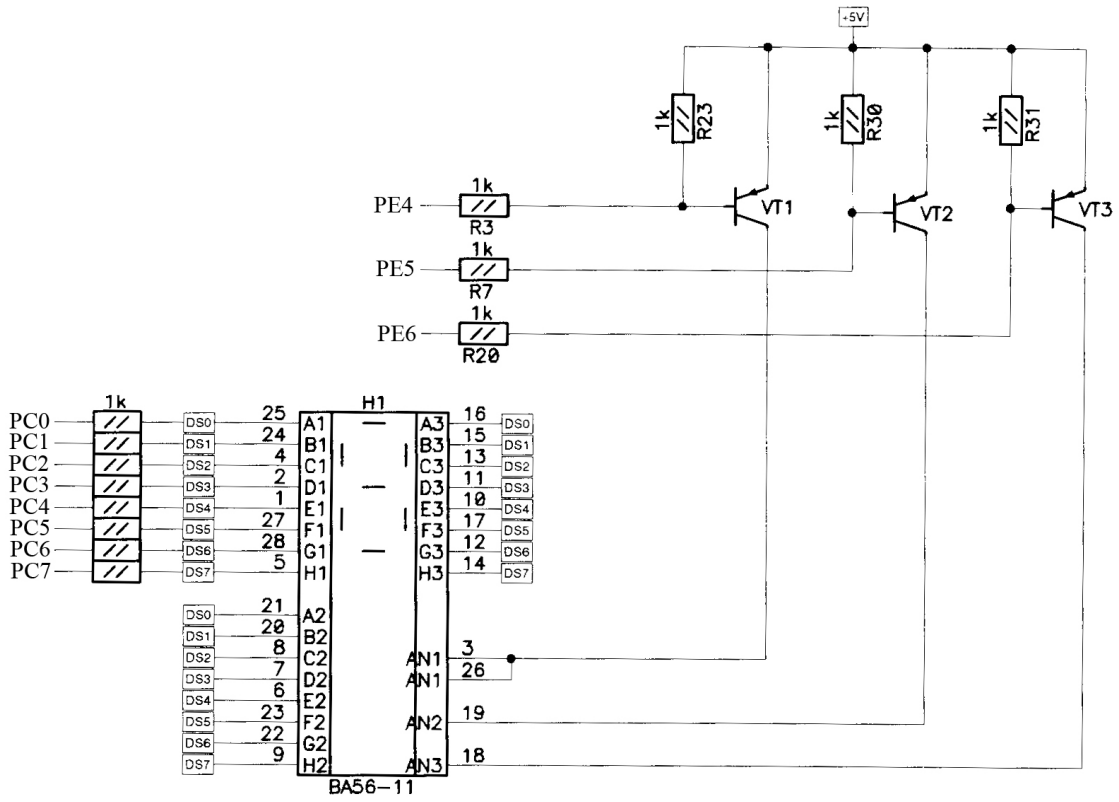


Рис. 4.1 Схема подключения светодиодного индикатора на плате лабораторного стенда ЛС-2.

Клавиатура является одним из наиболее распространенных устройств для ввода оператором в цифровую систему команд и данных. Обычно клавиши клавиатуры организованы в виде матрицы, и их состояние читается через параллельные порты.

На плате лабораторного макета смонтирована простая 12-кнопочная клавиатура, которая подключена к выводам PD0 – PD6 порта PD микроконтроллера. Клавиатура имеет вид матрицы 4 х 3, кнопки обозначены A1 – A3, B1 – B3, C1 – C3, D1 – D3, где цифры 1, 2, 3 определяют номер столбца, а буквы A, B, C, D указывают позицию кнопки в столбце. Подключение кнопок клавиатуры к выводам порта PD показано на рисунке 4.2. Для контроля состояния кнопки необходимо подать низкий потенциал (логический «0») на один из выводов микроконтроллера PD0, PD1, PD2 или PD3. В этом случае на выводе PD4, PD5 или PD6, подключенном к нажатой кнопке, установится «0».

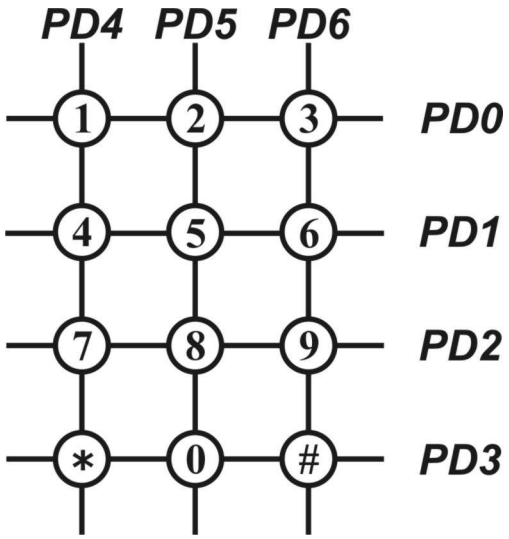


Рис. 4.1 Схема подключения клавиатуры на плате лабораторного стенда ЛС-2.

Для определения текущего состояния клавиатуры микроконтроллер должен периодически считывать состояния выводов PD4 – PD6, последовательно устанавливая на выводах PD0 – PD3 состояние "0". Для этого служит процедура опроса клавиатуры, посредством выполнения которой микроконтроллер определяет позицию нажатой кнопки. В реальных системах опрос клавиатуры производится периодически по сигналам таймера или осуществляется по запросу прерывания, формируемому клавиатурой. Основной проблемой при обслуживании клавиатуры является защита от дребезга контактов, который может привести к формированию ложной информации. Для защиты применяются как аппаратные, так и программные средства. На плате лабораторного макета аппаратных средств защиты от дребезга контактов нет, поэтому должны применяться исключительно программные методы, все варианты которых сводятся к повторному чтению состояния клавиатуры.

Нажатие клавиши должно вызывать ввод в цифровую систему определенного 8-разрядного символа. Кодировка клавиш, которая должна быть реализована при выполнении данной работы на лабораторном макете, задается преподавателем.

Порты ввода-вывода.

У микроконтроллеров ATmega128 имеется 6 универсальных параллельных портов ввода/вывода A, B, C, D, E, F и G.

Все порты ввода-вывода микроконтроллера работают по принципу чтение-модификация-запись при использовании их в качестве портов универсального ввода-вывода. Это означает, что изменение направления ввода-вывода одной линии порта командами SBI и CBI будет, происходит без ложных изменений направления ввода-вывода других линий порта. Данное распространяется также и на изменение логического уровня (если линия порта настроена на вывод) или на включение/отключение подтягивающих резисторов (если линия настроена на ввод). Каждый

выходной буфер имеет симметричную характеристику управления с высоким втекающим и вытекающим выходными токами. Выходной драйвер обладает нагрузочной способностью, которая позволяет непосредственно управлять светодиодными индикаторами. Ко всем линиям портов может быть подключен индивидуальный выборочный подтягивающий к плюсу питания резистор, сопротивление которого не зависит от напряжения питания. На всех линиях порта установлены защитные диоды, которые подключены к VCC и Общему (GND).

Регистры портов ввода-вывода.

Ссылки на регистры и биты регистров даны в общей форме. При этом символ “x” заменяет наименование порта A, B, C, D, E, F и G, а символ “n” заменяет номер разряда порта от 0 до 7.

Для каждого порта ввода-вывода в памяти ввода-вывода зарезервировано три ячейки: одна под регистр данных – PORTx, другая под регистр направления данных – DDRx и третья под состояние входов порта – PINx. Ячейка, хранящая состояние на входах портов, доступна только для чтения, а регистры данных и направления данных имеют двунаправленный доступ. Кроме того, установка бита выключения подтягивающих резисторов PUD регистра SFIOR отключает функцию подтягивания на всех выводах всех портов.

Регистр данных порта PORTx

	7	6	5	4	3	2	1	0
	PORTx7	**PORTx6**	**PORTx5**	**PORTx4**	**PORTx3**	**PORTx2**	**PORTx1**	**PORTx0**
Исходное значение	0	0	0	0	0	0	0	0

Регистр направления данных порта - DDRx

	7	6	5	4	3	2	1	0
	DDx7	**DDx6**	**DDx5**	**DDx4**	**DDx3**	**DDx2**	**DDx1**	**DDx0**
Исходное значение	0	0	0	0	0	0	0	0

Регистр входных данных порта - PINx

	7	6	5	4	3	2	1	0
	PINx7	**PINx6**	**PINx5**	**PINx4**	**PINx3**	**PINx2**	**PINx1**	**PINx0**
Исходное значение	0	0	0	0	0	0	0	0

Таблица 4.1 – Адреса регистров портов ввода-вывода.

ПОРТ / РЕГИСТР	A	B	C	D	E	F	G
PINx	$19($39)	$16($36)	$13($33)	$10($30)	$01($21)	$00($20)	($63)
DDRx	$1A($3A)	$17($37)	$14($34)	$11($31)	$02($22)	($61)	($64)
PORTx	$1B($3B)	$18($38)	$15($35)	$12($32)	$03($23)	($62)	($65)

Настройка выводов.

Режим и состояние для каждого вывода определяется значением соответствующих разрядов трех регистров: DDxn, PORTxn и PINxn.

Биты DDxn регистра DDRx определяют направленность линии ввода-вывода. Если DDxn=1, то Pxn конфигурируется на вывод. Если DDxn=0, то Pxn конфигурируется на ввод.

Если PORTxn = 1 при конфигурации линии порта на ввод, то разрешается подключение подтягивающего резистора. Для выключения данного резистора необходимо записать в PORTxn лог. 0 или настроить линию порта на вывод. Во время сброса все линии портов находятся в третьем (высокоимпедансном) состоянии.

Таблица 4.2 – Настройка вывода порта.

DDxn	PORTxn	PUD (в SFIOR)	Ввод-вывод	Подтягивающий резистор	Комментарий
0	0	X	Ввод	Нет	Третье состояние (Z-состояние)
0	1	0	Ввод	Да	Pxn будет источником тока при подаче внешнего низкого уровня
0	1	1	Ввод	Нет	Третье состояние (Z-состояние)
1	0	X	Вывод	Нет	Вывод лог. 0 (втекающий ток)
1	1	X	Вывод	Нет	Вывод лог. 1 (вытекающий ток)

ВЫПОЛНЕНИЕ ОСНОВНОГО ЗАДАНИЯ

1. Ознакомиться со схемой платы лабораторного стенда и подключением клавиатуры и индикатора к выводам микроконтроллера.
2. Запустить на персональном компьютере интегрированную систему программирования AVRStudio. Командой Project|New Project создать новый проект Lesson4. С использованием редактора текста создать демонстрационную программу:

```
.device ATmega128
.include      "m128def.inc"
;=====================================================================
;Определения регистров и битов
;=====================================================================
.def  temp    =r16
.def  dig1    =r17
.def  dig2    =r18
.def  dig3    =r19
.def  deley   =r20
.def  deley1  =r21
.def  scan_kbd=r22

.equ  razr1   =PE4
.equ  razr2   =PE5
.equ  razr3   =PE6
.equ  seg     =PORTC
.equ  kbd     =PIND
.equ  row1    =PD0
.equ  row2    =PD1
.equ  row3    =PD2
.equ  row4    =PD3
.equ  col1    =PD4
.equ  col2    =PD5
.equ  col3    =PD6
;=====================================================================
;********* Вектор сброса *********************
;=====================================================================
.CSEG
.org  $0000   ;начальный адрес программы
      rjmp    RESET
;=====================================================================
;Вывод показаний на индикатор из регистров dig1, dig2 и dig3
;=====================================================================
Display:
      mov     temp,dig1
      rcall   _rd_tbl
```

```
      cbi     PORTE,razr1
      rcall   _deley
      sbi     PORTE,razr1

      mov     temp,dig2
      rcall   _rd_tbl
      cbi     PORTE,razr2
      rcall   _deley
      sbi     PORTE,razr2

      mov     temp,dig3
      rcall   _rd_tbl
      cbi     PORTE,razr3
      rcall   _deley
      sbi     PORTE,razr3

      ret
;=====================================================================
;Подпрограмма перекодировки значения в семисегментный код
;=====================================================================
_rd_tbl:
      ldi     ZH,High(seg_table*2) ;загружаем указатель на таблицу
      ldi     ZL,Low(seg_table*2)
      add     ZL,temp              ;вычисляем адрес
      lpm                          ;читаем байт в r0
      out     seg,r0               ;включаем нужные сегменты
      ret
;=====================================================================
;Подпрограмма сканирования клавиатуры
;=====================================================================
Keyboard:
      cbi     PORTD,row1
      rcall   del_key
      in      temp,kbd
      sbi     PORTD,row1
      com     temp
      cbr     temp,$8F
      breq    _row2
      sbr     temp,(1<<row1)
      rjmp    _ret_kbd
_row2:
      cbi     PORTD,row2
      rcall   del_key
      in      temp,kbd
      sbi     PORTD,row2
      com     temp
      cbr     temp,$8F
      breq    _row3
      sbr     temp,(1<<row2)
      rjmp    _ret_kbd
_row3:
      cbi     PORTD,row3
      rcall   del_key
      in      temp,kbd
      sbi     PORTD,row3
      com     temp
      cbr     temp,$8F
      breq    _row4
```

```
      sbr     temp,(1<<row3)
      rjmp    _ret_kbd
_row4:
      cbi     PORTD,row4
      rcall   del_key
      in      temp,kbd
      sbi     PORTD,row4
      com     temp
      cbr     temp,$8F
      breq    _no_key
      sbr     temp,(1<<row4)
_ret_kbd:
      cp      temp,scan_kbd
      breq    _no_key
      mov     scan_kbd,temp
      rcall   key_det
      mov     dig1,dig2
      mov     dig2,dig3
      mov     dig3,temp
      ret
_no_key:
      ret
;=====================================================================
;Подпрограмма определения нажатой клавиши
;=====================================================================
key_det:
      cpi     temp,0b00010001
      brne    _key2
      ldi     temp,1
      ret
_key2:
      cpi     temp,0b00100001
      brne    _key3
      ldi     temp,2
      ret
_key3:
      cpi     temp,0b01000001
      brne    _key4
      ldi     temp,3
      ret
_key4:
      cpi     temp,0b00010010
      brne    _key5
      ldi     temp,4
      ret
_key5:
      cpi     temp,0b00100010
      brne    _key6
      ldi     temp,5
      ret
_key6:
      cpi     temp,0b01000010
      brne    _key7
      ldi     temp,6
      ret
_key7:
      cpi     temp,0b00010100
      brne    _key8
```

```
      ldi     temp,7
      ret
_key8:
      cpi     temp,0b00100100
      brne    _key9
      ldi     temp,8
      ret
_key9:
      cpi     temp,0b01000100
      brne    _key10
      ldi     temp,9
      ret
_key10:
      cpi     temp,0b00011000
      brne    _key11
      ldi     temp,10
      ret
_key11:
      cpi     temp,0b00101000
      brne    _key12
      ldi     temp,11
      ret
_key12:
      cpi     temp,0b01001000
      brne    _err_key
      ldi     temp,12
_err_key:
      ret
;=====================================================================
;Подпрограмма задержки для клавиатуры
;=====================================================================
del_key:
      ldi     deley,5
_del_key1:
      dec     deley
      brne    _del_key1
      ret
;=====================================================================
;Подпрограмма задержки для индикатора
;=====================================================================
_deley:
      ldi     deley1,1
_deley1:
      dec     deley
      brne    _deley1
      dec     deley1
      brne    _deley1
      ret
;=====================================================================
RESET:
;инициализация стека
      ldi     temp,High(RAMEND)
      out     SPH,temp              ;устанавливаем указатель стека
      ldi     temp,Low(RAMEND)      ;на конечный адрес ОЗУ контроллера
      out     SPL,temp
;очищаем регистры
      clr     dig1
      clr     dig2
```

```
      clr     dig3
      clr     scan_kbd
;настройка выходных линий разрядов и их выключение
      in      temp,DDRE
      sbr     temp,(1<<razr1)+(1<<razr2)+(1<<razr3)
      out     DDRE,temp
      in      temp,PORTE
      sbr     temp,(1<<razr1)+(1<<razr2)+(1<<razr3)
      out     PORTE,temp
;настройка выходных линий сегментов и их выключение
      ldi     temp,$FF
      out     DDRC,temp
      out     seg,temp
;настройка выходных линий клавиатуры
      in      temp,DDRD
      sbr     temp,(1<<row1)+(1<<row2)+(1<<row3)+(1<<row4)
      out     DDRD,temp
      ldi     temp,$FF
      out     PORTD,temp
;=====================================================================
;Основная программа
;=====================================================================
main:
      rcall   Display
      rcall   Keyboard
      rjmp    main
;=====================================================================
;Таблица констант в памяти программ
;=====================================================================
seg_table:
.db $C0,$F9,$A4,$B0,$99,$92,$82,$F8,$80,$90,$88,$83,$C6,$A1,$86,$8E,$FF
.EXIT
```

Программа Lesson4 определяет факт нажатия одной из клавиш 12-кнопочной клавиатуры, организованной в виде матрицы 4 строки x 3 столбца, заносит номер нажатой клавиши в регистр scan_kbd и отображает номер нажатой клавиши в шестнадцатеричном формате на семисегментном индикаторе макета. Подразумевается, что в каждый момент времени может быть нажата только одна клавиша. Код нажатой клавиши заносится в регистр temp и по нему определяется номер клавиши в подпрограмме key_det (это удобно при дальнейшем анализе и выполнении операций управления), который выводится на индикатор.

3. Выполнить команду Project/Build для компиляции проекта.
4. Загрузить программу lesson4.hex в микроконтроллер по методике пункта 2.4 «ЗАПИСЬ ПРОГРАММЫ В МИКРОКОНТРОЛЛЕР».

Задания

1. Составить программу, которая анализирует номер нажатой клавиши и формирует соответствующий десятичный номер на двух семисегментных индикаторах. (Кодировка клавиш задается преподавателем).

2. Разработать и реализовать различные варианты подавления дребезга контактов клавиатуры.

3. Составить программу, осуществляющую считывание трех последовательно вводимых с клавиатуры чисел и их вывод на три семисегментных индикатора.

4. Составить программу опроса клавиатуры, имеющую защиту от одновременного нажатия двух и более клавиш.

5. Составить программу опроса клавиатуры, которая рассматривает некоторые клавиши как служебные, изменяющие действие нажатых основных клавиш.

6. Составить программу опроса клавиатуры, которая наращивает значение в ячейке памяти, соответствующей клавише, если клавиша нажата и удерживается. Результат должен наблюдаться на семисегментных индикаторах.

7. Провести отладку разработанных программ на лабораторном стенде с использованием интегрированной системы программирования. Проверить правильность выполнения программ на макете, продемонстрировать результаты преподавателю.

РАБОТА №5

ЦИФРОВЫЕ СИСТЕМЫ НА БАЗЕ МИКРОКОНТРОЛЛЕРА ATmega128: реализация таймерных функций

Цель работы: изучение функционирования таймеров/счетчиков микроконтроллера ATmega128, получение практических навыков программирования микроконтроллерных систем для работы в реальном масштабе времени.

Введение

Микроконтроллер ATmega128 имеет четыре таймера/счетчика: два 8-разрядных Таймер 0, Таймер 2 и два 16-разрядных Таймер1, Таймер 3. Таймер 0, в дополнение к обычному режиму, может тактироваться асинхронно от внешнего генератора. Этот генератор оптимизирован под использование кварцевого кристалла на частоту 32768 кГц, что позволяет использовать Таймер 0 как часы реального времени (Real Time Clock - RTC).

8-разрядные таймеры/счетчики T/C0 и T/C2.

Таймеры/счетчики T/C0 и T/C2 предназначены для отсчета и измерения временных интервалов, генерации сигналов с широтно-импульсной модуляцией (ШИМ), а также как счетчик внешних событий.

Таймеры/счетчики T/C0 и T/C2 отличаются только наличием асинхронного режима работы в таймере/счетчике T/C0.

В состав таймеров/счетчиков входят 3 регистра ввода/вывода:

- счетный регистр TCNT0 (TCNT2)
- регистр управления TCCR0 (TCCR2)
- регистр сравнения OCR0 (OCR2)
- регистр ASSR в таймере/счетчике T/C0 для управления в асинхронном режиме.

В регистре флагов прерывания таймеров/счетчиков TIFR хранятся различные флаги состояния (переполнения, совпадения при сравнении и захвата события). Разрешение и запрещение прерываний производится посредством регистра масок прерываний таймеров/счетчиков TIMSK.

Точность и разрешение 8-разрядных таймеров/счетчиков растет с уменьшением коэффициента предварительного деления. Высокий коэффициент предварительного деления удобно использовать при реализации медленных операций или точной синхронизации редко происходящих действий.

Оба таймера/счетчика поддерживают две функции сравнения выхода, используя регистры сравнения выходов OCR0 и OCR2 как источники данных, сравниваемых с содержимым таймеров/счетчиков. В функции сравнения выхода входит очистка счетчика при совпадении, а также формирование, при совпадении, активных сигналов на выводах PB4(OC0/PWM0) и PB7(OC2/PWM2).

Таймеры/счетчики 0 и 2 можно использовать как 8-разрядные широтно-импульсные модуляторы. В этом режиме таймер/счетчик, совместно с регистром совпадения выхода, работает как автономный ШИМ с центрированными импульсами без ложных выбросов.

Регистр состояния асинхронного режима таймера/счетчика T/C0 - ASSR

Разряд	7	6	5	4	3	2	1	0	
$30 ($50)	**-**	**-**	**-**	**-**	**AS0**	**TCN0UB**	**OCR0UB**	**TCR0UB**	**TCCR0**
Исходное значение	0	0	0	0	0	0	0	0	

Разряд 4-7 - Зарезервированные биты

*Разряд 3 – **AS0**:* Разрешение асинхронного режима таймера/счетчика T/C0. При установленном (= 1) бите на вход предделителя таймера/счетчика 0 поступают импульсы с внешнего кварцевого

генератора. В этом режиме выводы TOSC1 и TOSC2 используются для подключения кварцевого резонатора.

*Разряд 2 – **TCN0UB**:* Состояние обновления регистра TCNT0

*Разряд 1 – **OCR0UB**:* Состояние обновления регистра OCRT0

*Разряд 0 – **TCR0UB**:* Состояние обновления регистра TCRT0

Регистр управления таймером/счетчиком - TCCR0

Разряд	7	6	5	4	3	2	1	0	
$33 ($53)	**FOC0**	**WGM00**	**COM01**	**COM00**	**WGM01**	**CS02**	**CS01**	**CS00**	**TCCR0**
Исходное значение	0	0	0	0	0	0	0	0	

Разряд	7	6	5	4	3	2	1	0	
$25 ($45)	**FOC2**	**WGM20**	**COM21**	**COM20**	**WGM21**	**CS22**	**CS21**	**CS20**	**TCCR2**
Исходное значение	0	0	0	0	0	0	0	0	

*Разряд 7 – **FOC0(FOC2):*** Принудительное изменение состояния вывода OC0(OC2).

*Разряд 6,3 – **WGM00,WGM01 (WGM20,WGM21)**:* Режим работы таймер/счетчика.

Таблица 5.1 – Выбор режима работы таймер/счетчика

Номер режима	WGM01 (WGM21)	WGM00 (WGM00)	Режим работы таймер/счетчика
0	0	0	Нормальный режим
1	0	1	ШИМ с коррекцией фазы
2	1	0	Сброс при совпадении
3	1	1	Быстрый ШИМ

*Разряд 5,4 – **COM01, COM00 (COM20, COM21)**:* Режим работы блока сравнения.

Таблица 5.2 – Поведение вывода OC0(OC2) в нормальном режиме и в режиме сброс при совпадении.

COM01 (COM21)	COM00 (COM20)	Описание
0	0	Таймер/счетчик отсоединен от выходного вывода OC0(OC2)
0	1	Состояние выходной линии OC0(OC2) меняется на противоположное
1	0	Сброс выходной линии OC0(OC2) (установка в состояние 0)
1	1	Установка выходной линии OC0(OC2) (установка в состояние 1)

Таблица 5.3 – Поведение вывода OC0(OC2) в ШИМ режимах.

COM01 (COM21)	COM00 (COM20)	Описание
0	0	Таймер/счетчик отсоединен от выходного вывода OC0(OC2)
0	1	Зарезервировано
1	0	Сброс выходной линии OC0(OC2) (установка в состояние 0)
1	1	Установка выходной линии OC0(OC2) (установка в состояние 1)

Разряд 2,1,0 – ***CS02, CS01, CS00 (CS22, CS21, CS20)****:* Выбор источника тактового сигнала.

Таблица 5.4. Выбор источника тактового сигнала для таймера/счетчика 0

CS02	CS01	CS00	Источник тактирования в зависимости от бита AS0 в регистре ASSR	
			AS0=0	AS0=1
0	0	0	таймер/счетчик T/C0 остановлен	
0	0	1	СК	TOSC1
0	1	0	СК / 8	TOSC1/8
0	1	1	СК / 32	TOSC1/32
1	0	0	СК / 64	TOSC1/64
1	0	1	СК / 128	TOSC1/128
1	1	0	СК / 256	TOSC1/256
1	1	1	СК / 1024	TOSC1/1024

где СК-частота тактирования микроконтроллера, TOSC1-частота внешнего тактового генератора.

Таблица 5.5. Выбор источника тактового сигнала для таймера/счетчика 2.

CS22	**CS21**	**CS20**	**Описание**
0	0	0	таймер/счетчик T/C2 остановлен
0	0	1	СК
0	1	0	СК / 8
0	1	1	СК / 64
1	0	0	СК / 256
1	0	1	СК / 1024
1	1	0	Внешний вывод T2 (PD7), спадающий фронт
1	1	1	Внешний вывод T2 (PD7), нарастающий фронт

Условие таймер/счетчик остановлен обеспечивает реализацию функции разрешения/запрещения таймера. Режим деления синхросигнала реализуется непосредственным делением тактовой частоты микроконтроллера СК. Если для тактирования Таймера/счетчика T/C2 используется внешний источник, то переключения на выводе PD7/(T2) будут воздействовать на счетчик, даже если этот вывод сконфигурирован как выход.

Счетный регистр таймера/счетчика 0 – TCNT0

Разряд	7	6	5	4	3	2	1	0	
$32 ($52)	**MSB**							**LSB**	**TCNT0**
Исходное значение	0	0	0	0	0	0	0	0	

Счетный регистр таймера/счетчика 2 - TCNT2

Разряд	7	6	5	4	3	2	1	0	
$24 ($44)	**MSB**							**LSB**	**TCNT2**
Исходное значение	0	0	0	0	0	0	0	0	

Эти два 8-разрядных регистра являются регистрами счета таймеров/счетчиков. Оба таймера/счетчика работают как инкрементирующие, либо как реверсивные (в ШИМ режиме) счетчики с возможностью чтения/записи. Если в таймер/счетчик записано некоторое значение и выбран источник тактового сигнала, то он продолжит счет с записанного значения с указанной тактовой частотой.

Регистр сравнения таймера/счетчика 0 – OCR0

Разряд	7	6	5	4	3	2	1	0	
$31 ($51)	**MSB**							**LSB**	**OCR0**
Исходное значение	0	0	0	0	0	0	0	0	

Регистр сравнения таймера/счетчика 2 – OCR2

Разряд	7	6	5	4	3	2	1	0	
$23 ($43)	**MSB**							**LSB**	**OCR2**
Исходное значение	0	0	0	0	0	0	0	0	

Регистры сравнения выходов являются 8-разрядными регистрами с возможностью чтения/записи. Выбор процедуры сравнения определяется регистрами TCCR0 и TCCR2. Совпадение при сравнении произойдет только тогда, когда таймер/счетчик досчитает до значения содержимого OCR. Программная запись одного и того же значения в таймер/счетчик и в регистр сравнения выхода не приведет к формированию совпадения при сравнении.

Совпадение при сравнении приведет к установке флага прерывания по совпадению в течение цикла процессора, следующего за событием.

16-разрядные таймеры/счетчики T/C1 и T/C3

Как и восьмиразрядные таймеры/счетчики T/C0 и T/C2, шестнадцатиразрядные таймер/счетчики T/C1 и T/C3 используются для формирования временных интервалов, подсчета внешних событий, формирования сигналов и генерации сигналов с ШИМ, а также в отличие от восьмиразрядных могут по внешнему сигналу сохранять свое текущее состояние в отдельном регистре ввода/вывода.

В состав каждого таймера/счетчика входят следующие регистры ввода/вывода:

- 16-разрядный счетный регистр TCNT1 (TCNT3);
- 16-разрядный регистр захвата ICR1 (ICR3);
- три 16-разрядных регистра сравнения OCR1A, OCR1B, OCR1C (OCR3A, OCR3B, OCR3C);
- три 8-разрядных регистра управления TCCR1A, TCCR1B, TCCR1C (TCCR3A, TCCR3B, TCCR3C).

Каждый из 16-разрядных регистров физически размещается в двух регистрах ввода/вывода, названия которых получаются добавлением к названию регистра буквы «H» (старший байт) и «L» (младший байт). Счетный регистр таймера счетчика TCNT1, например, размещается в регистрах TCNT1H:TCNT1L.

Адреса всех регистров таймеров/счетчиков T1 и T3 указаны в табл. 5.6.

Таблица 5.6. Адреса регистров таймеров/счетчиков T/C1 и T/C3.

	TCCRxA	TCCRxB	TCCRxC	TCNTx	OCRxA	OCRxB	OCRxC	ICRx
T/C1(x=1)	$2F($4F)	$2E($4E)	($7A)	$2D:$2C ($4D:$4C)	$2B:$2A ($4B:$4A)	$29:$28 ($49:$48)	($79:$78)	$27:$26 ($47:$46)
T/C3(x=3)	($8B)	($8A)	($8C)	($89:$88)	($87:$86)	($85:$84)	($83:$82)	($81:$80)

Регистр управления А таймеров/счетчиков.

	7	6	5	4	3	2	1	0	
	COMxA1	**COMxA0**	**COMxB1**	**COMxB0**	**COMxC1**	**COMxC0**	**WGMx1**	**WGMx0**	**TCCRxA**
Исходное значение	0	0	0	0	0	0	0	0	

Разряд 7,6 - ***COMxA1, COMxA0:*** Режим работы блока сравнения А.
Разряд 5,4 - ***COMxB1, COMxB0:*** Режим работы блока сравнения В.
Разряд 3,2 - ***COMxC1, COMxC0:*** Режим работы блока сравнения С.
Назначение разрядов 2-7 зависит от режима работы таймер/счетчика.
Разряд 1,0 - ***WGMx1, WGMx0:*** Режим работы таймер/счетчика.
Вместе с разрядами WGMx3:WGMx2 регистра TCCRxB определяют режим работы таймер/счетчика.

Регистр управления В таймеров/счетчиков.

	7	6	5	4	3	2	1	0	
	ICNCx	**ICESx**	**-**	**WGMx3**	**WGMx2**	**CSx2**	**CSx1**	**CSx0**	**TCCRxB**
Исходное значение	0	0	0	0	0	0	0	0	

Разряд 7 - ***ICNCx:*** Управление схемой подавления помех блока захвата.
Разряд 6 - ***ICESx:*** Выбор активного фронта сигнала захвата.
Разряд 5 - Зарезервирован.
Разряд 4,3 – ***WGMx3, WGMx2:*** Режим работы таймер/счетчика.
Вместе с разрядами WGMx1:WGMx0 регистра TCCRxA определяют режим работы таймер/счетчика.
Разряд 2,1,0 – ***CSx2, CSx1, CSx0:*** Управление тактовым сигналом таймер/счетчика.

Таблица 5.7. Выбор источника тактового сигнала для таймера/счетчика.

CSx2	CSx1	CSx0	Описание
0	0	0	таймер/счетчик остановлен
0	0	1	СК
0	1	0	СК / 8
0	1	1	СК / 64
1	0	0	СК / 256
1	0	1	СК / 1024
1	1	0	Внешний вывод Tx, спадающий фронт
1	1	1	Внешний вывод Tx, нарастающий фронт

Регистр управления С таймеров/счетчиков.

	7	6	5	4	3	2	1	0	
	FOCxA	**FOCxB**	**FOCxC**	**-**	**-**	**-**	**-**	**-**	**TCCRxC**
Исходное значение	0	0	0	0	0	0	0	0	

Разряд 7 - ***FOCxA:*** Принудительное изменение состояния вывода OCxA.
Разряд 6 - ***FOCxB:*** Принудительное изменение состояния вывода OCxA.
Разряд 5 - ***FOCxC:*** Принудительное изменение состояния вывода OCxA.
Разряд 4,3,2,1,0 - Зарезервирован.

Таблица 5.8 Режимы работы 16 разрядных таймер/счетчиков.

Номер режима	WGMx3	WGMx2	WGMxl	WGMx0	Режим работы таймера/счетчика	Величина, до которой производится счет
0	0	0	0	0	Нормальный	$FFFF
1	0	0	0	1	8-разрядный ШИМ с коррекцией фазы	$00FF
2	0	0	1	0	9-разрядный ШИМ с коррекцией фазы	S01FF
3	0	0	1	1	10-разрядный ШИМ с коррекцией фазы	$03FF
4	0	1	0	0	Сброс при совпадении	OCRxA
5	0	1	0	1	8-разрядный быстрый ШИМ	$00FF
6	0	1	1	0	9-разрядный быстрый ШИМ	$01FF
7	0	1	1	1	10-разрядный быстрый ШИМ	$03FF
8	1	0	0	0	ШИМ с коррекцией фазы и частоты	ICRn
9	1	0	0	1	ШИМ с коррекцией фазы и частоты	OCRxA
10	1	0	1	0	ШИМ с коррекцией фазы	ICRx
11	1	0	1	1	ШИМ с коррекцией фазы	OCRxA
12	1	1	0	0	Сброс при совпадении	ICRx
13	1	1	0	1	Зарезервировано	-
14	1	1	1	0	Быстрый ШИМ	ICRx
15	1	1	1	1	Быстрый ШИМ	OCRxA

Счетный регистр таймеров/счетчиков – TCNTx.

	7	6	5	4	3	2	1	0	
	MSB								**TCNTxH**
								LSB	**TCNTxL**
Исходное	0	0	0	0	0	0	0	0	
значение	0	0	0	0	0	0	0	0	

Этот 16-разрядный регистр содержит текущее значение 16-разрядного таймера/счетчика.

Регистры сравнения А таймеров/счетчиков - OCRxA

	7	6	5	4	3	2	1	0	
	MSB								**OCRxAH**
								LSB	**OCRxAL**
Исходное	0	0	0	0	0	0	0	0	
значение	0	0	0	0	0	0	0	0	

Регистры сравнения B таймеров/счетчиков - OCRxB

	7	6	5	4	3	2	1	0	
	MSB								**OCRxBH**
								LSB	**OCRxBL**
Исходное	0	0	0	0	0	0	0	0	
значение	0	0	0	0	0	0	0	0	

Регистры сравнения C таймеров/счетчиков - OCRxC

	7	6	5	4	3	2	1	0	
	MSB								**OCRxCH**
								LSB	**OCRxCL**
Исходное	0	0	0	0	0	0	0	0	
значение	0	0	0	0	0	0	0	0	

16-разрядные регистры сравнения выхода обеспечивают и чтение и запись.

Регистры сравнения таймера/счетчика хранят эталон, постоянно сравниваемый с состоянием таймера/счетчика. Действие, запускаемое совпадением при сравнении, определяется режимом работы блока сравнения таймера/счетчика. Совпадение при сравнении может произойти только, если таймер/счетчик досчитает до значения содержимого OCR.

Совпадение при сравнении устанавливает флаг прерывания в тактовом цикле процессора, следующем за самим совпадением.

Регистр захвата таймеров/счетчиков – ICRx.

	7	6	5	4	3	2	1	0	
	MSB								**ICRxH**
								LSB	**ICRxL**
Исходное	0	0	0	0	0	0	0	0	
значение	0	0	0	0	0	0	0	0	

16-разрядный регистр захвата входа обеспечивает только чтение содержимого.

При обнаружении на выводе ICx нарастающего или спадающего фронта сигнала (определяемого установкой ICESx в регистре TCCRxB) текущее состояние таймера/счетчика (значение регистра TCNTx) пересылается в регистр ICRx. Одновременно устанавливается флаг ICFx.

Обращение к 16-разрядным регистрам.

Каждый 16-разрядный регистр таймеров/счетчиков физически размещается в двух 8-разрядных регистрах. Соответственно для обращения к ним требуется выполнить по две операции чтения или записи. Для того чтобы запись или чтение обоих байт содержимого 16-разрядного регистра происходила одновременно, в составе каждого таймера/счетчика имеется специальный 8-разрядный регистр TEMP, предназначенный для хранения старшего байта значения (этот регистр используется только процессором и программно недоступен).

Для выполнения цикла записи 16-разрядного регистра первым должен быть загружен старший байт, который помещается в регистр TEMP. При последующей записи младшего байта он объединяется с содержимым регистра TEMP, и оба байта одновременно (в одном и том же машинном цикле) записываются в 16-разрядный регистр. Если требуется изменить несколько 16-разрядных регистров таймера/счетчика, а старшие байты всех записываемых значений одинаковы, загрузку старшего байта достаточно выполнить только один раз.

Для выполнения цикла чтения 16-разрядного регистра первым должен быть прочитан младший байт. При его чтении содержимое старшего байта помещается в регистр TEMP. При последующем чтении старшего байта возвращается значение, сохраненное в регистре TEMP.

Исключение составляют только регистры сравнения OCR1A/B/C (OCR3A/B/C), при чтении которых регистр TEMP не задействуется.

При выполнении цикла обращения к 16-разрядному регистру таймера/счетчика прерывания должны быть запрещены. В противном случае, если прерывание произойдет между двумя командами обращения к 16-разрядному регистру, а в подпрограмме обработки этого прерывания тоже будет произведено обращение к какому-либо из 16-разрядных регистров того же таймера/счетчика, содержимое регистра TEMP будет изменено. Как следствие, результат обращения к 16-разрядному регистру в основной программе будет неверным.

ВЫПОЛНЕНИЕ ОСНОВНОГО ЗАДАНИЯ

5. Ознакомиться со схемой платы лабораторного стенда и подключением клавиатуры и индикатора к выводам микроконтроллера.
6. Запустить на персональном компьютере интегрированную систему программирования AVRStudio. Командой Project|New Project создать новый проект Lesson4. С использованием редактора текста создать демонстрационную программу:

```
.device ATmega128
.include   "m128def.inc"
;=====================================================================
;Определения регистров и битов
;=====================================================================
.def  temp       =r16
.def  dig1       =r17
.def  dig2       =r18
.def  dig3       =r19
.def  deley      =r20
.def  beep       =r21

.equ  razr1      =PE4
.equ  razr2      =PE5
.equ  razr3      =PE6
.equ  seg        =PORTC
;*****таблица векторов прерываний***********************************
.CSEG
.org  $0000      ;начальный адрес программы
      rjmp  Start;переход к рабочей части программы (вектор Reset)
.org  $0002
      reti       ;Внешнее прерывание Int0
.org  $0004
      reti       ;Внешнее прерывание Int1
.org  $0006
      reti       ;Внешнее прерывание Int2
.org  $0008
      reti       ;Внешнее прерывание Int3
.org  $000A
      reti       ;Внешнее прерывание Int4
.org  $000C
      reti       ;Внешнее прерывание Int5
.org  $000E
      reti       ;Внешнее прерывание Int6
.org  $0010
      reti       ;Внешнее прерывание Int7
.org  $0012
      rjmp  comp2;Совпадение таймера/счетчика T2
```

```
.org $0014
     rjmp ovf2 ;Переполнение таймера/счетчика T2
.org $0016
     reti      ;Захват таймера/счетчика T1
.org $0018
     rjmp comp1a     ;Совпадение "A" таймера/счетчика T1
.org $001A
     reti      ;Совпадение "B" таймера/счетчика T1
.org $001C
     reti      ;Переполнение таймера/счетчика T1
.org $001E
     reti      ;Совпадение таймера/счетчика T0
.org $0020
     rjmp ovf0 ;Переполнение таймера/счетчика T0
.org $0022
     reti      ;Передача по SPI завершена
.org $0024
     reti      ;USART0, прием завершен
.org $0026
     reti      ;Регистр данных USART0 пуст
.org $0028
     reti      ;USART0, передача завершена
.org $002A
     reti      ;Преобразование АЦП завершено
.org $002C
     reti      ;Запись в EEPROM завершена
.org $002E
     reti      ;Аналоговый компаратор
.org $0030
     reti      ;Совпадение "C" таймера/счетчика T1
.org $0032
     reti      ;Захват таймера/счетчика T3
.org $0034
     reti      ;Совпадение "A" таймера/счетчика T3
.org $0036
     reti      ;Совпадение "B" таймера/счетчика T3
.org $0038
     reti      ;Совпадение "C" таймера/счетчика T3
.org $003A
     rjmp ovf3 ;Переполнение таймера/счетчика T3
.org $003C
     reti      ;USART1, прием завершен
.org $003E
     reti      ;Регистр данных USART1 пуст
.org $0040
     reti      ;USART1, передача завершена
.org $0042
     reti      ;Прерывание от модуля TWI
.org $0044
     reti      ;Готовность SPM
;***** Обработка прерывания Таймера0 *********************************
ovf0:
     inc  dig3
     cpi  dig3,10
     brne ovf_ret
     clr  dig3
     inc  dig2
     cpi  dig2,10
```

```
      brne  ovf_ret
      clr   dig2
      inc   dig1
      cpi   dig1,10
      brne  ovf_ret
      clr   dig1
ovf_ret:
      reti
;***** Обработка прерывания Таймера1 сравнение**********************
comp1a:
;инверсия линии выхода на пьезоизлучатель
      sbis  PORTD,PD7
      rjmp  comp2_1
      cbi   PORTD,PD7
      reti
comp2_1:
      sbi   PORTD,PD7
      reti
;***** Обработка прерывания Таймера2 ********************************
ovf2:
;включение светодиода
      cbi   PORTE,PE7
      reti
;***** Обработка прерывания Таймера2 сравнение**********************
comp2:
;выключение светодиода
      sbi   PORTE,PE7
      reti
;***** Обработка прерывания Таймера3 ********************************
ovf3:
      inc   beep
      inc   beep
      out   OCR1AL,beep ;изменение частоты генерируемого сигнала
      out   OCR2,beep   ;изменение яркости светодиода
      reti
;***** Основная программа*******************************************
Start:
;инициализация стека
      ldi   temp,High(RAMEND)
      out   SPH,temp
      ldi   temp,Low(RAMEND)
      out   SPL,temp
;очищаем регистры
      ldi   dig1,0
      ldi   dig2,0
      ldi   dig3,0
      ldi   beep,0
;настройка линии выхода на пьезоилучатель
      sbi   DDRD,PD7
      cbi   PORTD,PD7
;настройка линии выхода на светодиод
      sbi   DDRE,PE7
      cbi   PORTE,PE7
;настройка выходных линий разрядов и их выключение
      in    temp,DDRE
      sbr   temp,(1<<razr1)|(1<<razr2)|(1<<razr3)
      out   DDRE,temp
      sbi   PORTE,razr1
```

```
      sbi   PORTE,razr2
      sbi   PORTE,razr3
;настройка выходных линий сегментов и их выключение
      ldi   temp,$FF
      out   DDRC,temp
      out   seg,temp

;настройка таймера 0 на работу в асинхронном режиме
      in    temp,ASSR
;установка бита AS0,тактирование таймера частотой 32768Гц
      sbr   temp,(1<<AS0)
      out   ASSR,temp  ;разрешение работы Таймера 0 в асинхронном режиме
      ldi   temp,(1<<CS00)|(1<<CS02)
      out   TCCR0,temp       ;запуск Таймера 0
;разрешение прерывания по переполнению Таймера 0
      in    temp,TIMSK
      sbr   temp,(1<<TOIE0)
      out   TIMSK,temp

;настройка таймера 1 для работы в режиме «Сброс при совпадении»
      ldi   temp,(1<<CS11)|(1<<WGM12)
      out   TCCR1B,temp      ;запуск Таймера 1
;разрешение прерывания по сравнению Таймер 1
      in    temp,TIMSK
      sbr   temp,(1<<OCIE1A)
      out   TIMSK,temp

;настройка таймера 2 для работы в режиме «Быстрый ШИМ»
      ldi   temp,(1<<CS20)|(1<<WGM21)|(1<<WGM20)
      out   TCCR2,temp       ;разрешение работы Таймера 2
;разрешение прерывания по переполнению и сравнению Таймер 2
      in    temp,TIMSK
      sbr   temp,(1<<OCIE2)|(1<<TOIE2)
      out   TIMSK,temp

;настройка таймера 3
      ldi   temp,(1<<CS30)
      sts   TCCR3B,temp      ;разрешение работы Таймера 3
;разрешение прерывания по переполнению Таймера 3
      sbr   temp,(1<<TOIE3)
      sts   ETIMSK,temp

      ldi   temp,1
      out   OCR1AH,temp
      ldi   temp,0
      out   OCR1AL,temp

      sei                    ;разрешение всех прерываний
loop:
      rcall display
      rjmp  loop
;=====================================================================
;Вывод показаний на индикатор из регистров dig1, dig2 и dig3
;=====================================================================
Display:
      mov   temp,dig1
      rcall _rd_tbl
      cbi   PORTE,razr1
```

```
      rcall _deley
      sbi   PORTE,razr1

      mov   temp,dig2
      rcall _rd_tbl
      cbi   PORTE,razr2
      rcall _deley
      sbi   PORTE,razr2

      mov   temp,dig3
      rcall _rd_tbl
      cbi   PORTE,razr3
      rcall _deley
      sbi   PORTE,razr3
      ret
;=====================================================================
;Подпрограмма перекодировки значения в семисегментный код
;=====================================================================
_rd_tbl:
      ldi   ZH,High(seg_table*2)  ;загружаем указатель на таблицу
      ldi   ZL,Low(seg_table*2)
      add   ZL,temp               ;вычисляем адрес
      lpm                         ;читаем байт в r0
      out   seg,r0                ;включаем нужные сегменты
      ret
;=====================================================================
;Подпрограмма задержки для индикатора
;=====================================================================
_deley:
      dec   deley
      brne  _deley
      ret
;=====================================================================
;Таблица констант в памяти программ
;=====================================================================
seg_table:
.db $C0,$F9,$A4,$B0,$99,$92,$82,$F8,$80,$90,$88,$83,$C6,$A1,$86,$8E,$FF
.EXIT
```

Таймеры-счетчики микроконтроллеров используются для формирования временной сетки (как последовательности равных интервалов времени), отдельных временных интервалов различной длительности, а также ШИМ сигналов (широтно-импульсная модуляция). Кроме того, таймеры-счетчики являются инструментом анализа событий на основе механизмов захвата-сравнения.

Программа lesson5 демонстрирует работу всех четырех таймеров в разных режимах. Таймер 0 работает в асинхронном режиме, тактируясь частотой 32768Гц. Предделитель делит эту частоту на 128. В результате прерывания от этого таймера поступают 1 раз в секунду. В подпрограмме обработки прерывания от таймера 0 значения регистров dig1,dig2,dig3 увеличиваются на 1 и потом подпрограммой Display выводятся на индикатор. Таймер 1 работает в режиме «Сброс при сравнении» и используется для формирования частотного сигнала излучаемого пьезодинамиком. Таймер 2 работает в режиме «Быстрый ШИМ». Сформированный ШИМ сигнал поступает на светодиод, что изменяет его яркость свечения. Таймер 3 работает в нормальном режиме и используется для изменения значения регистра сравнения таймера 1 (от $100 до $1FF), что в свою очередь изменяет частоту генерируемого сигнала пьезодинамика. Также таймер 3 изменяет значение ШИМ сигнала генерируемого таймером 2(от $00 до $FF) , что приводит к изменению яркости свечения светодиода.

7. Выполнить команду Project/Build для компиляции проекта.
8. Загрузить программу lesson5.hex в микроконтроллер по методике пункта 2.4 «ЗАПИСЬ ПРОГРАММЫ В МИКРОКОНТРОЛЛЕР».

Задания

1. Составить программу, реализующую мигание светодиода на плате (перебор сегментов любого индикатора) с заданной частотой и длительностью. Изменение частоты (3-4 значения) производить нажатием определенных кнопок клавиатуры.
2. Составить программу, обеспечивающую периодическую выдачу на семисегментные индикаторы определенной последовательности чисел (4-5 значений). Период смены кадров (3-4 различных значения) задавать нажатием определенных кнопок клавиатуры.
3. Составить программу, реализующую подачу звуковых сигналов с периодически меняющейся тональностью с помощью пьезоизлучателя. Запуск и останов звучания производить нажатием определенных кнопок клавиатуры.
4. Составить программу, реализующую функции секундомера с индикацией на семисегментных индикаторах минут, секунд и десятых долей секунд. Запуск и останов секундомера производить нажатием определенных кнопок клавиатуры.

Провести отладку разработанных программ на лабораторном макете с использованием интегрированной системы программирования. Проверить правильность выполнения программ в реальном масштабе времени, продемонстрировать результаты работы преподавателю.

РАБОТА №6

ЦИФРОВЫЕ СИСТЕМЫ НА БАЗЕ МИКРОКОНТРОЛЛЕРА ATmega128: организация последовательного обмена данными

Цель работы: изучение функционирования последовательных портов микроконтроллера ATmega128, получение практических навыков программирования микроконтроллера для реализация последовательного обмена.

Введение

Микроконтроллер ATmega128 имеет два последовательных порта: USART0 и USART1.

Универсальный синхронный и асинхронный приемопередатчик USART предназначен для организации последовательной связи.

Отличительные особенности:

- Полнодуплексная работа (раздельные регистры последовательного приема и передачи)
- Асинхронная или синхронная работа
- Ведущее или подчиненное тактирование связи в синхронном режиме работы
- Высокая разрешающая способность генератора скорости связи
- Поддержка формата передаваемых данных с 5, 6, 7, 8 или 9 битами данных и 1 или 2 стоп-битами
- Аппаратная генерация и проверка бита паритета (четность/нечетность)
- Определение переполнения данных
- Определение ошибки в структуре посылки
- Фильтрация шума с детектированием ложного старт-бита и цифровым ФНЧ
- Три раздельных прерывания по завершении передачи, освобождении регистра передаваемых данных и завершении приема
- Режим многопроцессорной связи
- Режим удвоения скорости связи в асинхронном режиме

Инициализация USART.

Перед началом сеанса связи необходимо выполнить инициализацию USART. Процесс инициализации обычно состоит из установки скорости связи, задания формата посылки и разрешения работы передатчика и приемника. Если используется управление связью по прерываниям, то во время инициализации необходимо, чтобы был сброшен флаг общего разрешения прерываний (т.е. необходимо запретить все прерывания).

Если необходимо выполнить повторную инициализацию USART, например, для изменения скорости связи или формата посылки, то необходимо убедиться, чтобы во время инициализации передача была приостановлена. Флаг TXCx (здесь и далее по тексту вместо ‘x’ нужно подставить 0 или 1 в соответствии с номером приемопередатчика USART0 или USART1) может использоваться для проверки завершения работы передатчика, а флаг RXCx - для проверки отсутствия в приемном буфере несчитанных данных. Обратите внимание, что при использовании флага TXCx он должен сбрасываться программно перед началом каждой передачи (перед записью в UDRx).

Передача данных - Передатчик USART.

Работа передатчика USART разрешается путем установки бита разрешения передачи (TXENx) в регистре UCSRxB. После разрешения, функция вывода TxD как обычного порта заменяется на функцию выхода последовательной передачи данных. Скорость связи, режим работы и формат посылки должны быть установлены однократно перед началом какой-либо передачи.

Начало передачи инициируется записью передаваемых данных в буфер передатчика. Буферизованные данные в буфере передатчика будут перемещены в сдвиговый регистр, после того как он будет готов к отправке новой посылки. Запись в сдвиговый регистр новых данных происходит в состоянии ожидания (когда передача завершена) или сразу после завершения передачи последнего стоп-бита предыдущей посылки. Если в сдвиговый регистр записаны новые данные, то начинается передача одной посылки на скорости, определенной в регистре скорости связи.

Флаги и прерывания передатчика.

Передатчик USART имеет два флага, которые индицируют его состояние: флаг освобождения регистра данных UDREx и флаг завершения передачи TXCx в регистре UCSRxA. Оба флага могут использоваться для генерации прерываний.

Флаг освобождения регистра данных UDREx индицирует готовность буфера передатчика принять новые данные. Данный бит устанавливается при освобождении передающего буфера и сбрасывается, когда буфер передатчика содержит данные для передачи.

Флаг завершения передачи TXCx принимает единичное значение, если вся посылка в сдвиговом регистре передатчика была полностью сдвинута и в буфере передатчика отсутствуют новые данные для передачи. Флаг TXCx автоматически сбрасывается при переходе на вектор обработки прерывания по завершению передачи или программно сбрасывается путем записи в него лог. 1.

Прием данных - Приемник USART.

Работа приемника USART разрешается, если записать лог. 1 в бит разрешения работы приемника RXENx в регистре UCSRxB. После разрешения работы приемника обычное назначение вывода RxD заменяется на вход последовательного ввода данных приемника USART. Скорость связи, режим работы и формат посылки должны быть установлены однократно перед началом выполнения приема данных.

Приемник начинает прием данных только после определения действительного старт-бита. Выборка следующих за старт-битом бит данных происходит с частотой равной скорости связи. После получения первого стоп-бита, т.е. когда последовательная посылка полностью принята и находится в сдвиговом регистре приемника, содержимое сдвигового регистра перемещается в приемный буфер. Приемный буфер считывается при чтении регистра ввода-вывода UDRx.

Флаг и прерывание по завершению приема.

Флаг завершения приема RXCx в регистре UCSRxA сигнализирует о наличии несчитанных данных в приемном буфере. Данный флаг равен 1, если имеются несчитанные данные, и равен 0, если буфер приемника свободен (т.е. не содержит каких-либо несчитанных данных). Если приемник отключается (RXEN = 0), то приемный буфер будет сброшен и флаг RXC примет нулевое значение.

Если установлен бит разрешения прерывания по завершению приема RXCIEx в регистре UCSRxB, то при установке флага RXCx программа переходит на вектор обработки данного прерывания (при условии, что активно общее разрешение прерываний).

Описание регистров USART.

Таблица 6.1 - Адреса используемых в приемопередатчиках UART0 и UART1 регистров.

	UDRx	UCSRxA	UCSRxB	UCSRxC	UBRRxH	UBRRxL
USART0 (x=0)	$0C ($2C)	$0B ($2B)	$0A ($2A)	($95)	($90)	$09 ($29)
USART1 (x=1)	($9C)	($9B)	($9A)	($9D)	($98)	($99)

Буферные регистры данных передатчика и приемника USART расположены по одному и тому же адресу в области ввода-вывода, обозначенной как регистр данных UDRx. Если выполнять запись по адресу регистра UDRx, то записываемые данные помещаются в буферный регистр данных передатчика. По аналогии, при чтении регистра UDRx извлекается содержимое буферного регистра данных приемника.

Запись в буфер передатчика можно выполнять, если установлен флаг UDREx в регистре UCSRxA. Данные записанные в UDRx при сброшенном флаге UDREx будут игнорированы передатчиком.

Регистр A управления и статуса USART - UCSRxA

Разряд	7	6	5	4	3	2	1	0	
	RXCx	TXCx	UDREx	FEx	DORx	UPEx	U2Xx	MPCMx	UCSRxA
Чтение/запись	Чт.	Чт./Зп.	Чт.	Чт.	Чт.	Чт.	Чт./Зп.	Чт./Зп.	
Исх. значение	0	0	0	0	0	0	0	0	

Разряд 7 - **RXCx**: Флаг завершения приема

Данный флаг устанавливается, если в приемном буфере содержатся несчитанные данные и сбрасывается, когда приемный буфер свободен (т.е., не содержит несчитанных данных).

Разряд 6 - **TXCx**: Флаг завершения передачи

Данный флаг устанавливается, если вся посылка из сдвигового регистра передатчика полностью передана и в передающем буфере UDRx нет новых данных для передачи.

Разряд 5 - **UDREx**: Флаг освобождения регистра данных USART

Флаг UDREx индицирует о готовности приемного буфера UDRx к приему новых данных.

Разряд 4 - **FEx**: Ошибка посылки

Данный бит устанавливается, если при приеме посылки, была определена ошибка в структуре посылки. Под ошибкой структуры в данном случае понимается нулевое значение первого стоп-бита в этой посылке. Флаг FEx принимает нулевое значение, если принятый стоп-бит имел правильное единичное значение.

Разряд 3 - **DORx**: Флаг переполнения данных

Данный бит устанавливает, если выявлено условие переполнения. Переполнение данных возникает, если заполнен приемный буфер.

Разряд 2 - **UPEx**: Ошибка паритета

Данный бит устанавливается, если принятые данные характеризуется ошибкой паритета, при условии, что был разрешен контроль паритета (UPMx1 = 1).

Разряд 1 - **U2Xx**: Удвоение скорости связи

Данный бит оказывает влияние только в асинхронном режиме связи. Запись в данный бит лог. 1 уменьшает в два раза значение коэффициента деления скорости связи с 16 до 8, тем самым, удваивая скорость передачи данных в асинхронном режиме.

Разряд 0 - **MPCMx**: Режим многопроцессорной связи

Данный бит разрешает режим многопроцессорной связи. Если в бит MPCMx записать лог. 1, то все входящие данные, принимаемые приемником USART, будут игнорироваться, если они не содержат адресной информации. Состояние бита MPCMx не влияет на работу передатчика.

Регистр B управления и статуса - UCSRxB

Разряд	7	6	5	4	3	2	1	0	
	RXCIEx	TXCIEx	UDRIEx	RXENx	TXENx	UCSZx2	RXB8x	TXB8x	UCSRxB
Чтение/запись	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт.	Чт./Зп.	
Исх. значение	0	0	0	0	0	0	0	0	

Разряд 7 - **RXCIEx**: Разрешение прерывания по завершению приема

Запись в данный бит лог. 1 разрешает прерывание по флагу RXCx.

Разряд 6 - **TXCIEx**: Разрешение прерывания по завершению передачи

Запись в данный бит лог. 1 разрешает прерывание по флагу TXCx.

Разряд 5 - **UDRIEx**: Разрешение прерывания по освобождению регистра данных USART

Установка данного флага разрешает прерывание по флагу UDREx.

Разряд 4 - **RXENx**: Разрешение работы приемника

Запись в данный бит лог. 1 приводит к разрешению работы приемника USART.

Разряд 3 - **TXENx**: Разрешение работы передатчика

Запись в данный бит лог. 1 разрешает работу передатчика USART.

Разряд 2 - **UCSZx2**: Формат данных

Бит UCSZx2 вместе с битами UCSZx1:0 в регистре UCSRxC задают количество бит данных в посылке, как для приемника, так и для передатчика.

Разряд 1 - **RXB8x**: Значение 8-ого разряда принятых данных

RXB8x содержит значение 9-го бита принятой посылки с 9-битным форматом. Данный бит необходимо считать прежде, чем будут считаны младшие 8 бит из регистра UDRx.

Разряд 0 - **TXB8x**: 8-ой разряд передаваемых данных

TXB8x содержит значение 9-ого бита данных для передачи посылки с 9-битным форматом. Данный бит необходимо записать перед тем, как будут записаны младшие разряды данных в UDRx.

Регистр C управления и статуса USART - UCSRxC

Разряд	7	6	5	4	3	2	1	0	
	-	UMSELx	UPMx1	UPMx0	USBSx	UCSZx1	UCSZx0	UCPOLx	UCSRxC
Чтение/запись	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	
Исх. значение	0	0	0	0	0	0	0	0	

Разряд 7 - Зарезервированные биты

Разряд 6 - **UMSELx**: Выбор режима USART

Данный бит позволяет переключаться между асинхронным (UMSELx=0) и синхронным (UMSELx=1) режимами последовательной связи.

Разряды 5:4 - **UPMx1:0**: Режим паритета

Данные бита разрешают и устанавливают тип генерируемого и контролируемого паритета. После разрешения паритета передатчик автоматически генерирует и передает бит паритета в каждой посылке. Приемник генерирует бит паритета для принятых данных и сравнивает его со значением принятого в этой посылке бита паритета, а по результату сравнения устанавливает флаг ошибки паритета UPEx в регистре UCSRxA.

Таблица 6.2. Установки бит UPMx

UPMx1	UPMx0	Режим паритета
0	0	Отключен
0	1	(Резерв)
1	0	Четность
1	1	Нечетность

Разряд 3 - **USBSx**: Выбор числа стоп-бит

Данный бит определяет, сколько стоповых бит вставляет передатчик при генерации посылки. Приемник игнорирует эту настройку.

Таблица 6.3. Установки бита USBSx

USBSn	Число стоп-бит
0	1 бит
1	2 бита

Разряды 2:1 - **UCSZx1:0**: Формат данных

Биты UCSZx1:0 вместе с UCSZx2 в регистре UCSRxB задают количество бит данных в посылке, как для приемника, так и для передатчика.

Таблица 6.4. Установки бит UCSZx

UCSZx2	UCSZx1	UCSZx0	Формат данных
0	0	0	5 бит
0	0	1	6 бит
0	1	0	7 бит
0	1	1	8 бит
1	0	0	Резерв
1	0	1	Резерв
1	1	0	Резерв
1	1	1	9 бит

Разряд 0 - **UCPOLx**: Полярность синхронизации

Данный бит используется только в синхронном режиме. Если используется асинхронный режим, то в данный бит необходимо записать лог. 0. В синхронном режиме бит UCPOLx определяет соотношение между выборкой входящих данных и обновлением передаваемых данных и сигналом тактирования синхронной связи (XCKx).

Таблица 6.5. Установки бит UCPOLx

UCPOLx	Изменение передаваемых данных на выходе TxDx	Выборка принимаемых данных на входе RxDx
0	Нарастающий фронт XCKx	Спадающий фронт XCKx
1	Спадающий фронт XCKx	Нарастающий фронт XCKx

Регистры скорости связи USART - UBRRxL и **UBRRxH**

Разряд	7	6	5	4	3	2	1	0	
	-	-	-	-	UBRRx[11:8]				UBRRnH
	UBRRx[7:0]								UBRRnL
Чтение/запись	Чт. Чт./Зп.	Чт. Чт./Зп.	Чт. Чт./Зп.	Чт. Чт./Зп.	Чт./Зп. Чт./Зп.	Чт./Зп. Чт./Зп.	Чт./Зп. Чт./Зп.	Чт./Зп. Чт./Зп.	
Исх. значение	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	

Разряды 15:12 -Зарезервированные разряды

Разряды 11:0 - **UBRRx11:0**: Регистр скорости связи USART

UBRR - 12-разр. регистр, который задает значение скорости связи USART. Регистр UBRRxH содержит 4 старших разряда, а UBRRxL 8 младших разрядов значения скорости USART. Если во время передачи или приема изменить скорость связи, то сеанс связи будет нарушен. Запись в регистр UBRRxL инициирует обновление скорости связи.

Таблица 6.7 - Выражения для вычисления установок регистра скорости связи

Режим работы	Выражение для вычисления скорости связи	Выражения для вычисления значения UBRR
Нормальный асинхронный режим (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Асинхронный режим с удвоением скорости (U2X=1)	$BAUD = \frac{f_{OSC}}{8(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Синхронный ведущим режим	$BAUD = \frac{f_{OSC}}{2(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

BAUD - скорость связи (в битах в секунду, бод)
f_{OSC} – тактовая частота микроконтроллера
UBRR - Содержимое регистров UBRRH и UBRRL (0 … 4095).

ВЫПОЛНЕНИЕ РАБОТЫ

7. Изучить функционирование последовательных портов микроконтроллера ATmega128.
8. Запустить на персональном компьютере интегрированную систему программирования AVRStudio. Командой Project|New Project создать новый проект Lesson6. С использованием редактора текста создать демонстрационную программу:

```
.device ATmega128
.include  "m128def.inc"

;переопределения регистров и бит
.def      temp        =r16
.def      u_in_data   =r17
.def      deley       =r18
.def      deley1      =r19
.def      scan_kbd    =r22
.def      deleyk      =r23
.equ      razr1       =PE4
.equ      razr2       =PE5
.equ      razr3       =PE6
.equ      seg         =PORTC
.equ      kbd         =PIND
.equ      row1        =PD0
.equ      row2        =PD1
.equ      row3        =PD2
.equ      row4        =PD3
.equ      col1        =PD4
.equ      col2        =PD5
.equ      col3        =PD6
;*****таблица векторов прерываний***********************************
.CSEG
.org      $0000             ;начальный адрес программы
          rjmp   Start      ;переход к основной части программы
.org      $0024
          rjmp   UART_in    ;USART0, прием завершен
.org      $0026
          reti              ;Регистр данных USART0 пуст
.org      $0028
```

```
          reti              ;USART0, передача завершена
;***** Прерывание по завершению приема ********************************
UART_in:
          in     u_in_data,UDR0
          reti
;*** Таблица констант в памяти программ ***************
seg_table:
.db $C0,$F9,$A4,$B0,$99,$92,$82,$F8,$80,$90,$88,$83,$C6,$A1,$86,$8E,$FF
;***** Основная программа*********************************************
Start:
          ldi    temp,High(RAMEND);инициализация стека
          out    SPH,temp              ;
          ldi    temp,Low(RAMEND);
          out    SPL,temp         ;
;настройка выходных линий разрядов и их выключение
          in     temp,DDRE
          sbr    temp,(1<<razr1)+(1<<razr2)
          out    DDRE,temp
          sbi    PORTE,razr1
          sbi    PORTE,razr2
;настройка выходных линий сегментов и их выключение
          ldi    temp,$FF
          out    DDRC,temp
          out    seg,temp
;настройка выходных линий клавиатуры
          in     temp,DDRD
          sbr    temp,(1<<row1)+(1<<row2)+(1<<row3)+(1<<row4)
          out    DDRD,temp
          sbi    PORTD,row1
          sbi    PORTD,row2
          sbi    PORTD,row3
          sbi    PORTD,row4
;устанавливаем скорость обмена 9600 для частоты кварца 6МГц
          clr    temp
          sts    UBRR0H,temp
          ldi    temp,38
          out    UBRR0L,temp
;Разрешение работы приемника и передатчика
          ldi    temp,(1<<RXEN0)|(1<<RXCIE0)|(1<<TXEN0)
          out    UCSR0B,r16
; Установка формата посылки: 8 бит данных, 1стоп-бита
          ldi    temp, (1<<UCSZ01)|(1<<UCSZ00)
          sts    UCSR0C,r16
          clr    u_in_data  ;очистка регистра
          sei               ;разрешение всех прерываний
loop:
          rcall  keyboard
          rcall  Display
          tst    scan_kbd
          breq   loop
          out    UDR0,scan_kbd
          rjmp   loop
;Подпрограмма вывода показаний на индикатор из регистра u_in_data
Display:
          mov    temp,u_in_data
          cbr    temp,$F0
          rcall  _rd_tbl
          cbi    PORTE,razr2
```

```
          rcall  _deley
          sbi    PORTE,razr2
          mov    temp,u_in_data
          swap   temp
          cbr    temp,$F0
          rcall  _rd_tbl
          cbi    PORTE,razr1
          rcall  _deley
          sbi    PORTE,razr1
          ret
;Подпрограмма перекодировки значения в семисегментный код
_rd_tbl:
          ldi    ZH,High(seg_table*2)  ;загружаем указатель на таблицу
          ldi    ZL,Low(seg_table*2)
          add    ZL,temp               ;вычисляем адрес
          lpm                          ;читаем байт в r0
          out    seg,r0                ;включаем нужные сегменты
          ret
;Подпрограмма задержки для индикатора
_deley:
          ldi    deley1,2
_deley1:
          dec    deley
          brne   _deley1
          dec    deley1
          brne   _deley1
          ret
;Подпрограмма сканирования клавиатуры
Keyboard:
          cbi    PORTD,row1
          rcall  del_key
          in     temp,kbd
          sbi    PORTD,row1
          swap   temp
          com    temp
          cbr    temp,$F8
          breq   _row2
          rcall  key_calc
          cpi    temp,0
          breq   _no_key
          mov    scan_kbd,temp
          rjmp   _ret_kbd
_row2:
          cbi    PORTD,row2
          rcall  del_key
          in     temp,kbd
          sbi    PORTD,row2
          swap   temp
          com    temp
          cbr    temp,$F8
          breq   _row3
          rcall  key_calc
          cpi    temp,0
          breq   _no_key
          inc    temp
          inc    temp
          inc    temp
          mov    scan_kbd,temp
```

```
          rjmp   _ret_kbd
_row3:
          cbi    PORTD,row3
          rcall  del_key
          in     temp,kbd
          sbi    PORTD,row2
          swap   temp
          com    temp
          cbr    temp,$F8
          breq   _row4
          rcall  key_calc
          cpi    temp,0
          breq   _no_key
          inc    temp
          inc    temp
          inc    temp
          inc    temp
          inc    temp
          inc    temp
          mov    scan_kbd,temp
          rjmp   _ret_kbd
_row4:
          cbi    PORTD,row4
          rcall  del_key
          in     temp,kbd
          sbi    PORTD,row4
          swap   temp
          com    temp
          cbr    temp,$F8
          breq   _no_key
          rcall  key_calc
          cpi    temp,0
          breq   _no_key
          inc    temp
          inc    temp
          inc    temp
          inc    temp
          inc    temp
          inc    temp
          inc    temp
          inc    temp
          inc    temp
          mov    scan_kbd,temp
          rjmp   _ret_kbd
_no_key:
          clr    scan_kbd
_ret_kbd:
          ret
;Подпрограмма определения нажатой клавиши в ряду
key_calc:
          cpi    temp,1
          brne   _key2
          ret
_key2:
          cpi    temp,2
          brne   _key3
          ret
_key3:
```

```
          cpi    temp,4
          brne   _err_key
          ldi    temp,3
          ret
_err_key:
          clr    temp
          ret
;Подпрограмма задержки для клавиатуры
del_key:
          ldi    deleyk,50
_del_key1:
          dec    deleyk
          brne   _del_key1
          ret
.EXIT
```

Эта программа демонстрирует работу универсального приемопередатчика USART микроконтроллера ATmega128. Последовательный обмен проходит со скоростью 9600 бит в сек. При приеме данных используется прерывание. Полученные данные отображаются на индикаторе в шестнадцатеричном формате.

9. Выполнить команду Project/Build для компиляции проекта, при получении сообщения об ошибке компиляции исправить ее и повторить этот пункт.
10. Соединить два лабораторных стенда, разъемы XP12, прилагаемым в комплекте кабелем (эти действия производить при выключенном питании стенда).
11. Включить питание стендов.
12. Загрузить программу lesson7.hex в каждый микроконтроллер по методике пункта 2.4 «ЗАПИСЬ ПРОГРАММЫ В МИКРОКОНТРОЛЛЕР».
13. После запуска программы в микроконтроллере, нажимая клавиши на клавиатуре стенда, убедится в передаче информации по USART. При правильной настройке параметров порта USART на индикаторе другого стенда будет отображаться код нажатой клавиши (см табл.6.8).

Таблица 6.8 – Соответствие кодов клавишам клавиатуры стенда.

Клавиша	1	2	3	4	5	6	7	8	9	*	0	#
Код	01	02	03	04	05	06	07	08	09	0A	0b	0C

Задания

1. Составить программы асинхронного приема и передачи данных с контролем принятого байта. При приеме заданных байт осуществлять включение или выключение светодиода.
2. Составить программы асинхронного приема и передачи данных на разных скоростях обмена. Посмотреть, как будет передаваться информация при настройке разной скорости передачи на двух соединенных стендах.

РАБОТА №7

ЦИФРОВЫЕ СИСТЕМЫ НА БАЗЕ МИКРОКОНТРОЛЛЕРА ATmega128: обслуживание аналогового компаратора

Цель работы: изучение функционирования встроенного аналогового компаратора микроконтроллера ATmega128.

Аналоговый компаратор.

Аналоговый компаратор сравнивает уровни напряжений на неинвертирующем входе AIN0(PE2) и инвертирующем входе AIN1(PE3). Если напряжение на неинвертирующем входе AIN0 превышает напряжение на инвертирующем входе AIN1, то выход аналогового компаратора ACO принимает единичное состояние. Выход компаратора может быть настроен для использования в качестве источника входного сигнала для схемы захвата фронтов таймера-счетчика 1. Кроме того, компаратор может генерировать собственный запрос на обработку прерывания. Пользователь может выбрать несколько событий, по которым возникает прерывание: нарастающий, падающий фронт на выходе компаратора или любое его изменение.

Регистр специальных функций ввода-вывода – SFIOR

Разряд	7	6	5	4	3	2	1	0	
	TSM	–	–	–	ACME	PUD	PSR0	PSR321	SFIOR
Чтение/Запись	Чт./Зп.	Чт.	Чт.	Чт.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	
Начальное знач.	0	0	0	0	0	0	0	0	

Разряд 3 – ACME: Выбор мультиплексора на входе аналогового компаратора

Если выключен аналогово-цифровой преобразователь (ADEN=0 в регистре ADCSRA) и в данный разряд записана лог. 1, то к инвертирующему входу аналогового компаратора подключен выход аналогового мультиплексора АЦП. Запись в данный разряд лог. 0 приведет к подключению инвертирующего входа аналогового компаратора к выводу микроконтроллера AIN1.

Регистр состояния и управления аналогового компаратора – ACSR

Разряд	7	6	5	4	3	2	1	0	
	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
Чтение/Запись	Чт./Зп.	Чт./Зп.	Чт.	Чт/Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	
Начальное знач.	0	0	x	0	0	0	0	0	

Разряд 7 – ACD: Отключение аналогового компаратора

Запись в данный разряд лог. 1 приводит к снятию питания с аналогового компаратора. Данный разряд можно устанавливать в любой момент при необходимости отключения аналогового компаратора. Его использование позволяет снизить энергопотребление в активном режиме и режиме холостого хода. Перед изменением бита ACD необходимо отключить прерывание по аналоговому компаратору путем сброса бита ACIE в регистре ACSR. В противном случае может возникнуть прерывание после изменения значения данного бита.

Разряд 6 – ACBG: Подключение источника опорного напряжения к аналоговому компаратору

После установки данного бита к неинвертирующему входу компаратора подключается источник опорного напряжения. После сброса данного разряда неинвертирующий вход компаратора связан с выводом AIN0 микроконтроллера.

Разряд 5 – ACO: Выход аналогового компаратора

Этот бит выхода аналогового компаратора связан непосредственно с выходом ACO.

Разряд 4 – ACI: Флаг прерывания аналогового компаратора

Данный разряд устанавливается аппаратно, при возникновении события в соответствии с установками бит ACIS1 и ACIS0. Запрос на обработку прерывания аналогового компаратора

выполняется, если установлены биты ACIE и I в регистре SREG. ACI сбрасывается аппаратно при переходе на соответствующий вектор обработки прерывания. Альтернативно, бит ACI можно сбросить программно путем записи лог. 1 в данный флаг.

Разряд 3 – ACIE: Разрешение прерывания аналогового компаратора

Если в данный разряд записана лог. 1 и установлен бит I в регистре статуса, то прерывание по аналоговому компаратору активизируется. Запись в данный разряд лог. 0 приводит к отключению данного прерывания.

Разряд 2 – ACIC: Подключение аналогового компаратора к схеме захвата фронтов

Установка данного разряда приводит к разрешению совместной работы схемы захвата фронтов таймера-счетчика 1 и аналогового компаратора. В этом случае, выход аналогового компаратора непосредственно подключен к входному каскаду схемы захвата фронтов, позволяя к компаратору добавить функции подавления шумов и настройки фронтов прерывания по захвату фронта таймером-счетчиком 1.После записи в данный разряд лог. 0 связь между аналоговым компаратором и схемой захвата фронтов разрывается. Для активизации прерывания схемы захвата фронтов таймера-счетчика 1 по срабатыванию аналогового компаратора необходимо установить бит TICIE1 в регистре маски прерывания таймера (TIMSK).

Разряды 1, 0 – ACIS1, ACIS0: Выбор события прерывания аналогового компаратора

Данные разряды определяют, какое событие приводит к генерации запроса на прерывание аналогового компаратора. Варианты установок данных разрядов и их назначение представлены в таблице 7.1.

Таблица 7.1 - Установки разрядов ACIS1, ACIS0

ACIS1	**ACIS0**	**Событие**
0	0	Прерывание по любому изменению на выходе компаратора
0	1	Зарезервировано
1	0	Прерывание по падающему фронту на выходе компаратора
1	1	Прерывание по нарастающему фронту на выходе компаратора

Перед изменением бит ACIS1/ACIS0 необходимо отключить прерывание по аналоговому компаратору путем сброса бита разрешения прерывания в регистре ACSR. В противном случае может возникнуть прерывание при изменении значений данных бит.

Мультиплексированный вход аналогового компаратора.

Имеется возможность использовать выводы ADC7..0 в качестве неинвертирующих входов аналогового компаратора. Для организации такого ввода используется мультиплексор АЦП и, следовательно, в этом случае АЦП должен быть отключен. Если установлен бит разрешения подключения мультиплексора к аналоговому компаратору (бит ACME в SFIOR) и выключен АЦП (ADEN=0 в регистре ADCSRA), то состояние разрядов MUX2..0 регистра ADMUX определяют, какой вывод микроконтроллера подключен к неинвертирующему входу аналогового компаратора (см. табл. 7.2). Если ACME сброшен или установлен ADEN, то в качестве неинвертирующего входа аналогового компаратора используется вывод микроконтроллера AIN1.

Таблица 7.2 – Мультиплексированный вход аналогового компаратора

ACME	**ADEN**	**MUX2..0**	**Неинвертирующий вход аналогового компаратора**
0	x	xxx	AIN1
1	1	xxx	AIN1
1	0	000	ADC0
1	0	001	ADC1
1	0	010	ADC2
1	0	011	ADC3
1	0	100	ADC4
1	0	101	ADC5
1	0	110	ADC6
1	0	111	ADC7

ВЫПОЛНЕНИЕ РАБОТЫ

1. Изучить функционирование блока аналогового компаратора микроконтроллера ATmega128 и схему подключения его входов в данном лабораторном стенде (см. рис. 7.1).

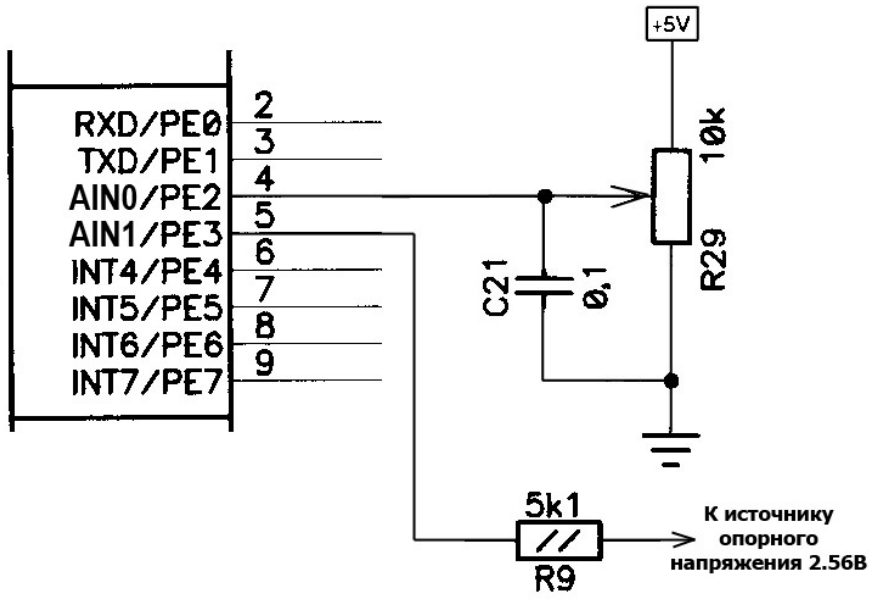


Рисунок 7.1 – Схема подключения аналогового компаратора.

2. Запустить на персональном компьютере интегрированную систему программирования AVRStudio. Командой Project|New Project создать новый проект Lesson7. С использованием редактора текста создать демонстрационную программу:

```
.device ATmega128
.include   "m128def.inc"
     in    R16,DDRE          ;
     sbr   R16,(1<<PE7)      ;установка линии порта PE7 на выход
     out   DDRE, R16         ;разрешение работы линии порта PE7 на выход
     sbi   PORTE,PE7         ;выключение светодиода
loop:
     in    R16,ACSR
     sbrs  R16,ACO     ;если выход компаратора ACO=1 включаем светодиод
     rjmp  comp_1      ;если ACO=0 выключаем светодиод
     cbi   PORTE,PE7   ;включение светодиода
     rjmp   loop
comp_1:
     sbi   PORTE,PE7   ;выключение светодиода
     rjmp   loop
```

Эта программа демонстрирует работу блока аналогового компаратора микроконтроллера ATmega128. В качестве индикатора выхода используется светодиод подключенный к порту PE7.

3. Выполнить команду Project/Build для компиляции проекта.

4. Если компиляция прошла без ошибок загрузить полученную программу lesson7.hex в микроконтроллер по методике пункта 2.4 «ЗАПИСЬ ПРОГРАММЫ В МИКРОКОНТРОЛЛЕР», при получении сообщения об ошибке компиляции исправить ее и повторить пункт 3.

5. После запуска программы в микроконтроллере, вращая ручку переменного резистора на стенде, наблюдать изменение состояния светодиода (включен\выключен). При равенстве напряжений на выводах AIN0 и AIN1 светодиод должен поменять свое состояние.

Задания

1. Написать программу, использующую прерывания от аналогового компаратора.
2. Написать программу, отображающую состояние выхода на одном разряде индикатора (0 или 1).
3. Написать программу, реагирующую на изменение фронта сигнала на выходе компаратора нарастающего или спадающего.

РАБОТА №8

ЦИФРОВЫЕ СИСТЕМЫ НА БАЗЕ МИКРОКОНТРОЛЛЕРА ATmega128: обслуживание АЦП

Цель работы: изучение функционирования встроенного АЦП микроконтроллера ATmega128.

Аналогово-цифровой преобразователь.

ATmega128 содержит 10-разр. АЦП последовательного приближения. АЦП связан с 8-канальным аналоговым мультиплексором, 8 однополярных входов которого связаны с линиями порта F. Входы АЦП могут объединятся попарно для ввода дифференциальных напряжений. Два дифференциальных входа (ADC1, ADC0 и ADC3, ADC2) содержат каскад со ступенчатым программируемым усилением: 0 дБ (1x), 20 дБ (10x), или 46 дБ (200x). Если выбрано усиление 1x или 10x, то можно ожидать 8-разр. разрешение, а если 200x, то 7-разрядное.

Принцип действия.

АЦП преобразовывает входное аналоговое напряжение в 10-разр. код методом последовательных приближений. Минимальное значение соответствует уровню GND, а максимальное уровню AREF равное 2,56В в данном лабораторном стенде.

Канал аналогового ввода и каскад дифференциального усиления выбираются путем записи битов MUX в регистре ADMUX. В качестве однополярного аналогового входа АЦП может быть выбран один из входов ADC0…ADC7, а также GND и выход фиксированного источника опорного напряжения 1,22 В.

Работа АЦП разрешается путем установки бита ADEN в ADCSRA. Выбор опорного источника и канала преобразования не возможно выполнить до установки ADEN.

АЦП генерирует 10-разрядный результат, который помещается в пару регистров данных АЦП ADCH и ADCL. По умолчанию результат преобразования размещается в младших 10-ти разрядах 16-разр. слова (выравнивание справа), но может быть опционально размещен в старших 10-ти разрядах (выравнивание слева) путем установки бита ADLAR в регистре ADMUX.

Практическая полезность представления результата с выравниванием слева существует, когда достаточно 8-разрядное разрешение, т.к. в этом случае необходимо считать только регистр ADCH. В другом же случае необходимо первым считать содержимое регистра ADCL, а затем ADCH, чем гарантируется, что оба байта являются результатом одного и того же преобразования. Как только выполнено чтение ADCL блокируется доступ к регистрам данных со стороны АЦП. Это означает, что если считан ADCL и преобразование завершается перед чтением регистра ADCH, то ни один из регистров не может модифицироваться и результат преобразования теряется. После чтения ADCH доступ к регистрам ADCH и ADCL со стороны АЦП снова разрешается.

Одиночное преобразование запускается путем записи лог. 1 в бит запуска преобразования АЦП ADSC. Данный бит остается в высоком состоянии в процессе преобразования и сбрасывается по завершении преобразования. Если в процессе преобразования переключается канал аналогового ввода, то АЦП автоматически завершит текущее преобразование прежде, чем переключит канал.

В режиме автоматического перезапуска АЦП непрерывно оцифровывает аналоговый сигнал и обновляет регистр данных АЦП. Данный режим задается путем записи лог. 1 в бит ADFR регистра ADCSRA. Первое преобразование инициируется путем записи лог. 1 в бит ADSC регистра

ADCSRA. В данном режиме АЦП выполняет последовательные преобразования, независимо от того сбрасывается флаг прерывания АЦП ADIF.

Регистры управления АЦП.

Регистр управления и статуса АЦП – **ADCSRA**.

Разряд	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	**ADCSRA**
Чтение/запись	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	
Исх. значение	0	0	0	0	0	0	0	0	

Разряд 7 – **ADEN**: Разрешение работы АЦП.

Запись в данный бит лог. 1 разрешает работу АЦП. Если в данный бит записать лог. 0, то АЦП отключается, даже если он находился в процессе преобразования.

Разряд 6 – **ADSC**: Запуск преобразования АЦП.

В режиме одиночного преобразования установка данного бита инициирует старт каждого преобразования. В режиме автоматического перезапуска установкой этого бита инициируется только первое преобразование, а все остальные выполняются автоматически. Первое преобразование после разрешения работы АЦП, инициированное битом ADSC, выполняется по расширенному алгоритму и длится 25 тактов синхронизации АЦП, вместо обычных 13 тактов. Это связано с необходимостью инициализации АЦП.

Разряд 5 – **ADFR**: Выбор режима автоматического перезапуска АЦП.

Если в данный бит записать лог. 1, то АЦП перейдет в режим автоматического перезапуска. В этом режиме АЦП автоматически выполняет преобразования и модифицирует регистры результата преобразования через фиксированные промежутки времени. Запись лог. 0 в этот бит прекращает работу в данном режиме.

Разряд 4 – **ADIF**: Флаг прерывания АЦП

Данный флаг устанавливается после завершения преобразования АЦП и обновления регистров данных. Если установлены биты ADIE и I (регистр SREG), то происходит прерывание по завершении преобразования. Флаг ADIF сбрасывается аппаратно при переходе на соответствующий вектор прерывания. Альтернативно флаг ADIF сбрасывается путем записи лог. 1 в него. Обратите внимание, что при выполнении команды "чтение-модификация-запись" с регистром ADCSRA ожидаемое прерывание может быть отключено. Данное также распространяется на использование инструкций SBI и CBI.

Разряд 3 – **ADIE**: Разрешение прерывания АЦП

После записи лог. 1 в этот бит, при условии, что установлен бит I в регистре SREG, разрешается прерывание по завершении преобразования АЦП.

Разряды 2:0 – **ADPS**2:0: Биты управления предделителем АЦП

Данные биты определяют на какое значение будет поделена тактовая частота ЦПУ перед подачей на вход синхронизации АЦП. Значения коэффициентов деления указаны в таблице 8.1.

Если требуется максимальная разрешающая способность (10 разрядов), то частота синхронизации должна быть в диапазоне 50…200 кГц. Если достаточно разрешение менее 10 разрядов, но требуется более высокая частота преобразования, то частота на входе АЦП может быть установлена свыше 200 кГц.

Таблица 8.1 – Управление предделителем АЦП

ADPS2	ADPS1	ADPS0	Коэффициент деления
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Регистр управления мультиплексором АЦП– **ADMUX**

Разряд	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	**ADMUX**
Чтение/запись	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	
Исх. значение	0	0	0	0	0	0	0	0	

Разряд 7:6 – **REFS**1:0: Биты выбора источника опорного напряжения

Данные биты определяют, какое напряжение будет использоваться в качестве опорного для АЦП. При работе с лабораторным стендом ЛС-2 нужно использовать ***только внешнее опорное напряжение*** (биты REFS0 и REFS1 должны быть равны 0).

Таблица 7.2 – Выбор опорного источника АЦП

REFS1	REFS0	Опорный источник
0	0	AREF, внутренний ИОН отключен
0	1	AVCC с внешним конденсатором на выводе AREF
1	0	Зарезервировано
1	1	Внутренний источник опорного напряжения 2.56В с внешним конденсатором на выводе AREF

Разряд 5 – **ADLAR**: Бит управления представлением результата преобразования

Бит ADLAR влияет на представление результата преобразования в паре регистров результата преобразования АЦП. Если ADLAR = 1, то результат преобразования будет иметь левосторонний формат, в противном случае - правосторонний. Действие бита ADLAR вступает в силу сразу после изменения, независимо от выполняющегося параллельно преобразования.

Разряд 4:0 – **MUX**4:0: Биты выбора аналогового канала и коэффициента усиления

Данные биты определяют, какие из имеющихся аналоговых входов подключаются к АЦП. Кроме того, с их помощью можно выбрать коэффициент усиления для дифференциальных каналов (см. табл. 8.2). Если значения бит изменить в процессе преобразования, то механизм их действия вступит в силу только после завершения текущего преобразования (после установки бита ADIF в регистре ADCSRA).

Таблица 8.2 – Выбор входного канала и коэффициента усиления

MUX4..0	**Однополярный вход**	**Неинвертирующий дифференциальный вход**	**Инвертирующий дифференциальный вход**	**Коэффициент усиления, Ку**
00000	ADC0	Нет		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	Нет	ADC0	ADC0	10
01001		ADC1	ADC0	10
01010		ADC0	ADC0	200
01011		ADC1	ADC0	200
01100		ADC2	ADC2	10
01101		ADC3	ADC2	10
01110		ADC2	ADC2	200
01111		ADC3	ADC2	200
10000		ADC0	ADC1	1
10001		ADC1	ADC1	1
10010		ADC2	ADC1	1
10011		ADC3	ADC1	1
10100		ADC4	ADC1	1
10101		ADC5	ADC1	1
10110		ADC6	ADC1	1
10111		ADC7	ADC1	1
11000		ADC0	ADC2	1
11001		ADC1	ADC2	1
11010		ADC2	ADC2	1
11011		ADC3	ADC2	1
11100		ADC4	ADC2	1
11101		ADC5	ADC2	1
11110	1.23В	Нет		
11111	0В(GND)			

Регистры результата преобразования – ADCL и **ADCH**

При ADLAR = 0:

Разряд	15	14	13	12	11	10	9	8	
	-	-	-	-	-	-	ADC9	ADC8	**ADCH**
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	**ADCL**
	7	6	5	4	3	2	1	0	
Чтение/запись	Чт.	Чт.	Чт.	Чт.	Чт.	Чт.	Чт.	Чт.	
	Чт.	Чт.	Чт.	Чт.	Чт.	Чт.	Чт.	Чт.	
Исх. значение	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

При ADLAR = 1:

Разряд	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	**ADCH**
	ADC1	ADC0	-	-	-	-	-	-	**ADCL**
	7	6	5	4	3	2	1	0	
Чтение/запись	Чт.	Чт.	Чт.	Чт.	Чт.	Чт.	Чт.	Чт.	
	Чт.	Чт.	Чт.	Чт.	Чт.	Чт.	Чт.	Чт.	
Исх. значение	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

По завершении преобразования результат помещается в этих двух регистрах. При использовании дифференциального режима преобразования результат представляется в коде двоичного дополнения.

Левосторонний формат ADLAR=1 представления результата удобно использовать, если достаточно 8 разрядов. В этом случае 8-разрядный результат хранится в регистре ADCH и, следовательно, чтение регистра ADCL можно не выполнять. При правостороннем формате **необходимо сначала считать ADCL, а затем ADCH**.

ВЫПОЛНЕНИЕ РАБОТЫ

1. Изучить функционирование блока АЦП микроконтроллера ATmega128 и схему подключения входов АЦП в данном лабораторном стенде (см. рис. 8.1). Обратить внимание, что входное напряжение, подаваемое на АЦП имеет размах от 0В до 5В, а опорное напряжение равно 2,56В.

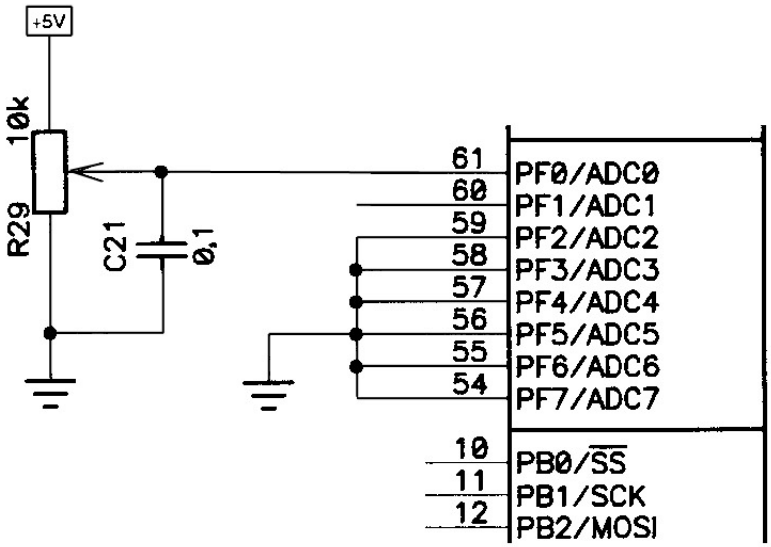


Рисунок 8.1 – Схема подключения переменного резистора к входу АЦП.

2. Запустить на персональном компьютере интегрированную систему программирования AVRStudio. Командой Project|New Project создать новый проект Lesson8. С использованием редактора текста создать демонстрационную программу:

```
.device ATmega128
.include   "m128def.inc"
;** таблица векторов прерываний ***************************
.CSEG
.org  $0000                ;начальный адрес программы
      rjmp  Start          ;переход к началу программы
.org  $002A
      rjmp  ADC_end        ;Преобразование АЦП завершено
;** Основная программа  ************************************
Start:
      ldi   R16,High(RAMEND);инициализация стека
      out   SPH,R16
      ldi   R16,Low(RAMEND) ;
      out   SPL,R16         ;
;** настройка первого разряда индикатора для вывода результата **
      in    R16,DDRE
      sbr   R16,(1<<PE4)    ;установка линии первого разряда на выход
      out   DDRE, R16       ;разрешение работы линии порта на выход

      sbr   R16,$FF         ;установка всех линий сегментов на выход
      out   DDRC, R16       ;разрешение работы линии порта на выход
      out   PORTC,R16       ;выключение всех сегментов
;*************************************************************
      in    R16,ADCSRA
      sbr   R16,(1<<ADEN)|(1<<ADIE)|(1<<ADPS2)|(1<<ADPS1)
      out   ADCSRA, R16          ;разрешение работы АЦП

      in    R16,ADMUX
```

```
      sbr   R16,(1<<ADLAR)  ;уст. левостороннего формата результата,
      out   ADMUX, R16      ;нулевого канала и внешнего ИОН
      sei                   ;разрешение всех прерываний
      sbi   ADCSRA,ADSC     ;запуск первого преобразования
loop:
      rjmp  loop            ;зацикливание    программы    до    прихода
прерывания
;***** Обработка прерывания от АЦП ******************************
ADC_end:
      in    R16,ADCH        ;считывание результата преобразования
      com   R16             ;инверсия для правильного отображения
      out   PORTC,R16       ;вывод на сегменты
;на   индикаторе   будет   отображаться   двоичный   код   результата
преобразования
      sbi   ADCSRA,ADSC     ;запуск следующего преобразования
      reti                  ;выход из прерывания
```

Эта программа демонстрирует настройку и обслуживание блока АЦП микроконтроллера ATmega128 для работы в режиме однократного преобразования по прерываниям с получением 8-ми битного результата и выводом ни первый разряд индикатора в двоичном виде.

3. Выполнить команду Project/Build для компиляции проекта.

4. Если компиляция прошла без ошибок загрузить полученную программу lesson8.hex в микроконтроллер по методике пункта 2.4. «ЗАПИСЬ ПРОГРАММЫ В МИКРОКОНТРОЛЛЕР», при получении сообщения об ошибке исправить ее и повторить пункт 3.

5. После запуска программы в микроконтроллере на индикаторе в первом разряде должен отобразиться результат преобразования АЦП в двоичном виде (см. табл. 8.3). Вращая ручку переменного резистора на стенде проконтролировать изменение показаний на индикаторе. Дать объяснение, почему показания изменяются не во всем диапазоне вращения ручки переменного резистора.

Таблица 8.3 – Соответствие сегментов индикатора битам в регистре результата ADCH.

Биты регистра ADCH	0	1	2	3	4	5	6	7
Сегменты индикатора	a	b	c	d	e	f	g	точка

Задания

4. Написать аналогичную программу, измеряющую напряжение на переменном резисторе по прерываниям таймера 1 с цикличностью задаваемой преподавателем.

5. Написать программу, отображающую округленный до 8 разрядов результат измерения в шестнадцатеричном виде на двух разрядах индикатора в диапазоне значений от 00 до FF.

6. Написать программу, отображающую округленный до 8 разрядов результат измерения в вольтах в диапазоне от 0.00 до 2.55В.

ПРИЛОЖЕНИЕ 1.

Схема электрическая принципиальная. Часть 1.

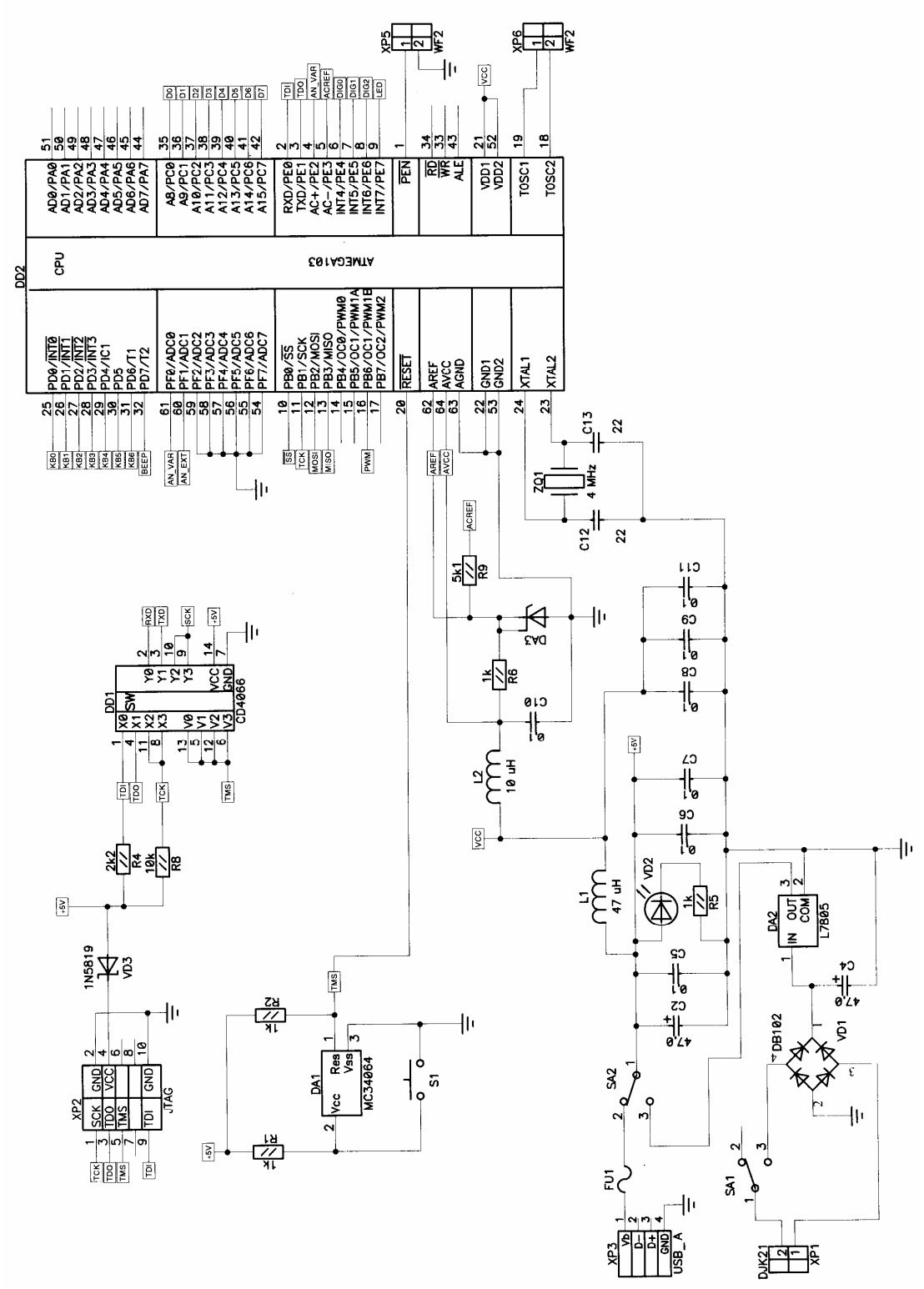
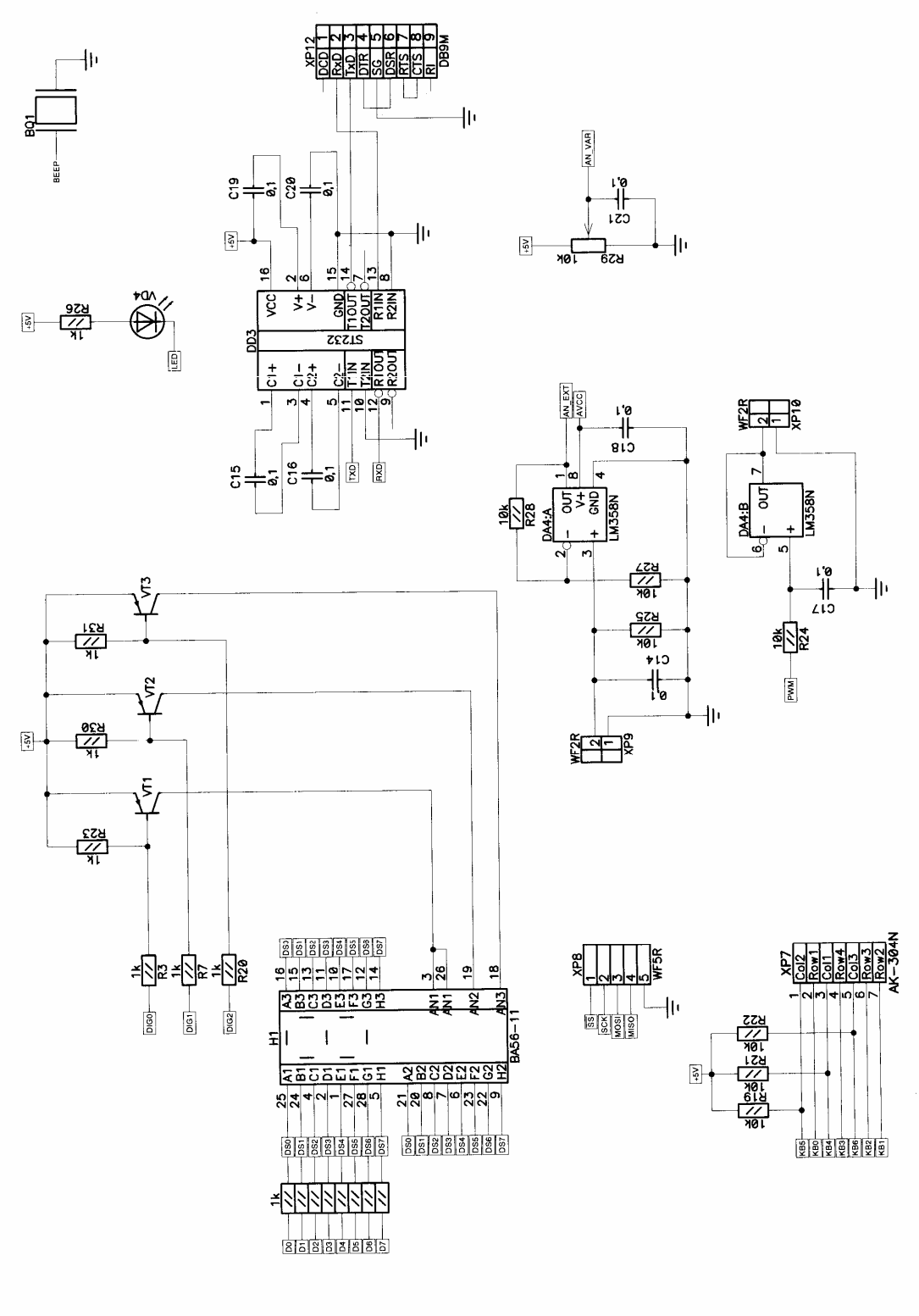


Схема электрическая принципиальная. Часть 2.



ПРИЛОЖЕНИЕ 2.

Схема расположения элементов на печатной плате.