

Министерство образования и науки Российской Федерации

Курский государственный университет

Кафедра *программного обеспечения и администрирования  
информационных систем*

Основы функционирования ЭВМ

Лабораторный практикум (Выпуск 2)

Курск 2021

**Основы функционирования ЭВМ (Выпуск 2):** методические указания к выполнению цикла лабораторных работ / сост. А. П. Жмакин; Курск. гос. ун-т. – Курск, 2021, 47 с.

Данное пособие можно рассматривать как продолжение (или альтернатива) методических указаний [2] и предназначено прежде всего для знакомства с принципами взаимодействия процессора с внешними устройствами ЭВМ.

В отличие от [2], в настоящем пособии используется программная модель учебной ЭВМ, названная нами fN8 – модель 8-разрядной двоичной ЭВМ с архитектурой фон Неймана. В состав модели, помимо процессора и памяти, включены внешние устройства (контроллер клавиатуры, символьный и графический дисплей, таймерные системы, контроллер семисегментной индикации и матричной клавиатуры), причём открытая архитектура и заложенный в модель резерв адресного пространства ввода/вывода позволяет расширять набор подключаемых ВУ.

Пособие предназначено для студентов направления 02.03.03 *Математическое обеспечение и администрирование информационных систем* (курс «Архитектура вычислительных систем»), а также может быть использовано студентами направлений 38.03.05 *Бизнес-информатика*, 01.04.02 *Прикладная математика и информатика* и 10.03.01 *Информационная безопасность* в курсе «Архитектура компьютеров».

# Содержание

<b>I</b>	<b>Описание модели ЭВМ</b>	<b>6</b>
<b>1.</b>	<b>Архитектура учебной ЭВМ fN8</b>	<b>6</b>
1.1.	Основные характеристики модели . . . . .	6
1.2.	Организация адресного пространства . . . . .	6
1.3.	Директивы компилятора . . . . .	8
1.4.	Ресурсы процессора . . . . .	9
<b>2.</b>	<b>Система команд</b>	<b>10</b>
2.1.	Система операций и форматы . . . . .	10
2.2.	Таблица команд . . . . .	12
2.3.	Пояснения к некоторым командам таблицы 1 . . . . .	16
2.3.1.	Безадресные команды десятичной коррекции . . . . .	16
2.3.2.	Умножение . . . . .	16
2.3.3.	Деление . . . . .	16
2.3.4.	Цикл . . . . .	17
2.3.5.	Команды безусловной передачи управления . . . . .	17
<b>3.</b>	<b>Внешние устройства</b>	<b>17</b>
3.1.	Диспетчер внешних устройств . . . . .	17
3.2.	Контроллер клавиатуры . . . . .	18
3.3.	Символьный дисплей . . . . .	20
3.4.	Таймер-2 . . . . .	21
3.4.1.	Режимы работы Таймера-2 . . . . .	21
3.4.2.	Формат и назначение разрядов регистра TSCR . . . . .	23
3.4.3.	Особенности организации доступа к 16-разрядным регистрам по 8-разрядному интерфейсу . . . . .	23
3.5.	Таймер-5 . . . . .	24
3.5.1.	Режимы работы . . . . .	26
3.5.2.	Форматы регистров состояния/управления . . . . .	26
3.5.3.	Формирование выходного сигнала . . . . .	27
3.5.4.	Буферирование старшего байта . . . . .	28
3.5.5.	Описание режимов работы . . . . .	28
3.6.	Контроллер матричной клавиатуры и 7-сегментной динамической инди- кации . . . . .	30
3.6.1.	Двухканальный цифровой осциллограф . . . . .	33
<b>II</b>	<b>Лабораторный практикум</b>	<b>34</b>
<b>4.</b>	<b>Лабораторная работа № 1.</b>	
	<b>Арифметические операции с многобайтовыми данными</b>	<b>34</b>
4.1.	Кодирование числовых данных . . . . .	34

4.2. Выполнение арифметических операций сложения и вычитания с «длинными» данными . . . . .	35
4.2.1. Упакованный формат . . . . .	35
4.2.2. Распакованный формат . . . . .	35
4.3. Задания . . . . .	36
4.4. Содержание отчёта . . . . .	37
<b>5. Лабораторная работа № 2.</b>	
<b>Программирование систем контроля времени</b>	<b>39</b>
5.1. Возможности таймерных систем микроконтроллеров и их программных моделей в fN8 . . . . .	39
5.2. Рекомендации по выполнению лабораторной работы . . . . .	41
5.3. Содержание отчёта . . . . .	43
5.4. Контрольные вопросы . . . . .	43
5.5. Варианты заданий . . . . .	45
<b>6. Лабораторная работа № 3.</b>	
<b>Широтно-импульсная модуляция</b>	<b>46</b>
<b>7. Лабораторная работа № 4.</b>	
<b>Работа с графическим дисплеем</b>	<b>46</b>

## Список сокращений

ВПС – верхний предел счёта;

ВУ – внешние устройства;

ОЗУ – оперативное запоминающее устройство;

РВУ – регистр(ы) внешних устройств;

РОН – регистр(ы) общего назначения;

ШИМ – широтно-импульсная модуляция;

ШИМ ФК – широтно-импульсная модуляция с фазовой коррекцией;

ЭВМ – электронная вычислительная машина;

## Часть I

# Описание модели ЭВМ

## 1. Архитектура учебной ЭВМ fN8

### 1.1. Основные характеристики модели

- архитектура фон Неймана (общее ОЗУ для программы и данных);
- формат ячеек – 1 байт;
- объём ОЗУ – 1024 байта;
- стек – в ОЗУ;
- число РОН – 8;
- адресное пространство ввода/вывода –  $2^7$  адресов;
- число векторов прерываний – 8.

Структурная схема fN8 показана на рис. 1.

### 1.2. Организация адресного пространства

В модели fN8 реализовано три адресных пространства:

- ОЗУ – 1024 байта;
- РОН – 8 байт;
- РВУ – 128 байт.

При проектировании форматов команд стремились обеспечить для всех команд одинаковую длину – 16 бит. В форматах команд ввода/вывода предусмотрено 7-битовое поле адреса, а в регистровых командах – 3-битовое (см. форматы 10, 11, 12 и 5 на рис. 2). Это позволяет адресовать в командах весь объём пространств ввода/вывода и регистров. Однако в форматах команд, адресующих ячейки ОЗУ (6, 7, 8, 9) поле адреса составляет всего 8 бит. Разрядности счётчика команд PCL и указателя стека SPL так же составляют 8 бит. Следовательно, в этом случае поле адреса таких команд и содержимое регистров PCL и SPL определяет только *смещение в соответствующем сегменте* пространства ОЗУ; очевидно, размер сегмента составляет 256 байт.

В адресном пространстве ОЗУ размещается четыре сегмента с адресами:

Сегмент 0 – 0x000 – 0x0FF

Сегмент 2 – 0x200 – 0x2FF

Сегмент 1 – 0x100 – 0x1FF

Сегмент 3 – 0x300 – 0x3FF

причём среди этих сегментов необходимо выделить сегменты кода, данных и стека.

Для управления размещением сегментов в процессоре предусмотрен специальный программно-доступный регистр SR, в котором SR[1:0] определяет номер сегмента кода (CS), SR[3:2] – номер сегмента данных (DS), SR[5:4] – номер сегмента стека (SS). По умолчанию CS = 00, DS = 10 и SS = 11, то есть под программу отводится 0 и 1 сегменты, под данные – сегмент 2, а под стек – сегмент 3.

# Процессор

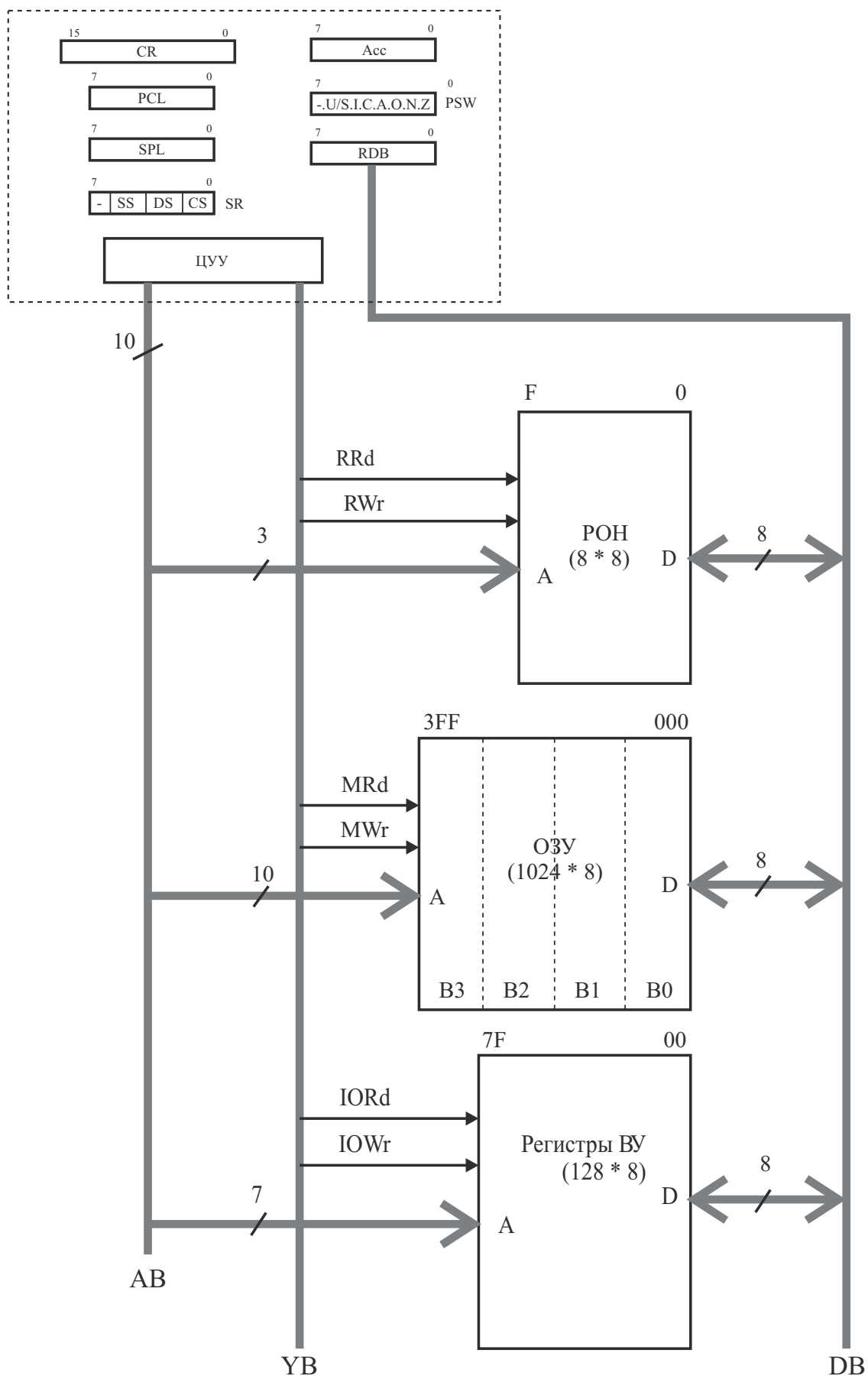


Рисунок 1. Структурная схема fN8

Таблица векторов прерываний размещается в сегменте 0 по адресам 0x00 .. 0x0F и может содержать до восьми векторов, причём *вектор 0 зарезервирован за RESET*. Каждый вектор занимает два байта и содержит 10-разрядный адрес обработчика прерывания. Значение вектора 0 по умолчанию (точка старта программы) – 0x0010.

Адресное пространство ввода/вывода составляет 128 байт, в нём может располагаться до 128 регистров внешних устройств<sup>1</sup> (ВУ). Разделение адресного пространства между подключаемыми ВУ осуществляется путём присвоения каждому устройству базового адреса, кратного 16. Таким образом, одновременно к системе можно подключить до 8 ВУ, каждое из которых может использовать до 16 адресов пространства ввода/вывода.

### 1.3. Директивы компилятора

При написании программы на языке Ассемблер можно пользоваться следующими директивами компилятора:

- .c <сегмент> – выбирает текущий сегмент компиляции;
- .org <адрес> – изменяет текущий адрес компиляции;
- .db <байт, байт, ... байт> – загрузка констант размером в 1 байт в текущий сегмент начиная с текущего адреса компиляции;
- .dw <слово, метка, ... слово> – загрузка констант размером в 2 байта в текущий сегмент начиная с текущего адреса компиляции.

В процессе компиляции возможна загрузка констант в сегмент данных (например, таблицы ASCII-кодов символов). По умолчанию сегменты «0» и «1» отведены для кода, сегмент «2» – для данных, а сегмент «3» – для стека. Если (лучше в конце текста программы) поставить директиву **.c 2**, то компилятор продолжит компиляцию фактически в сегмент данных. Например, если требуется разместить таблицу 7-сегментных кодов<sup>2</sup> десятичных цифр в сегменте данных начиная с адреса 0x90, то в конце текста программы можно добавить такой фрагмент:

```
.c 2
.org 0x90
.db 0x3F, 0x06, 0x5B, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F, 0x77
```

Использование этих директив позволяет изменить установленную по умолчанию точку старта программы – 0x10, например, на адрес 0x20 (a). Ещё проще поставить в нужном месте метку, например, *Start:* и объявить её точкой старта (b).

a)	b)
.org 0	.org 0
.db 0x20, 0x0	.dw Start

Эти же директивы позволяют автоматизировать заполнение таблицы векторов прерываний. Таблица векторов прерываний в fN8 размещается в младших адресах памяти

<sup>1</sup>При соответствующей организации схемы ВУ допускается присваивание одинакового адреса двум различным регистрам, из которых один является регистром ввода, а другой – регистром вывода

<sup>2</sup>В этом примере сегменту **A** индикатора соответствует младший бит кода.



0x000 – 0x00F, каждый вектор занимает два байта. Вектор 0 определяет точку старта (по умолчанию — 0x010), вектора 1, 2, 3 и 4 по умолчанию присваиваются клавиатуре, таймеру-2, таймеру-5 и контроллеру 7-сегментной индикации соответственно. Остальные вектора (5 — 7) можно использовать для других разрабатываемых внешних устройств или для подключения нескольких экземпляров одинаковых ВУ с разными векторами. Адрес младшего байта вектора определяется как его удвоенный номер.

Если поставить метки в начале всех обработчиков прерываний, то загрузка векторов прерываний может выглядеть следующим образом (для случая, когда вектора определяются последовательно):

```
.org 2
.dw IntKey, IntTim2, IntTim5, Int7Seg
```

По адресам неиспользуемых векторов в таблице векторов прерываний сохраняются значения 0. Если в системе используются те ВУ, вектора прерываний которых по умолчанию следуют не подряд, то можно или при подключении поменять им вектора в желаемом порядке или в директиве заполнения таблицы по неиспользуемым адресам записывать 0, например:

```
.org 2
.dw IntKey, 0, 0, Int7Seg ,
```

если подключать только контроллер клавиатуры и контроллер 7-сегментной индикации и оставить им назначаемые по умолчанию вектора.

## 1.4. Ресурсы процессора

Вне адресных пространств – в процессоре – расположены программно-доступные объекты:

- Асс[7:0] – аккумулятор;
- PCL[7:0] – счётчик команд;
- SPL[7:0] – указатель стека;
- SR[7:0] – сегментный регистр;
- PSW[7:0] – регистр слова состояния процессора;

Регистр слова состояния процессора поддерживает следующие флаги:

PSW[0] = Z – нулевой результат;  
 PSW[1] = N – отрицательный результат;  
 PSW[2] = O – арифметическое переполнение;  
 PSW[3] = AC – дополнительный перенос;  
 PSW[4] = C – перенос;  
 PSW[5] = I – разрешение прерывания;

Объекты процессора CR[15:0] – регистр команд и DR[7:0] – регистр данных не являются программно-доступными и отображают код выполняемой команды и значение второго операнда бинарной операции АЛЮ соответственно.

## 2. Система команд

### 2.1. Система операций и форматы

Система команд включает в себя следующие операции: арифметические и логические операции над аккумулятором (Асс) и ячейкой памяти или регистром с размещением результата в Асс, команды управления битами, команды пересылки, ввода/вывода, передачи управления (включая вызовы подпрограмм), управление прерываниями.

Адресация в командах с ячейками ОЗУ – прямая и непосредственная. В регистровых командах – прямая, косвенная и несколько вариантов автоиндексной адресации. Адресация в командах ввода/вывода – только прямая. Кроме того, возможна адресация отдельных битов в любой ячейке сегмента данных ОЗУ или в любом РВУ.

Все команды имеют размер 16 бит. Большинство команд при этом являются одноадресными.

Форматы всех команд fN8 показаны на рис. [2](#)

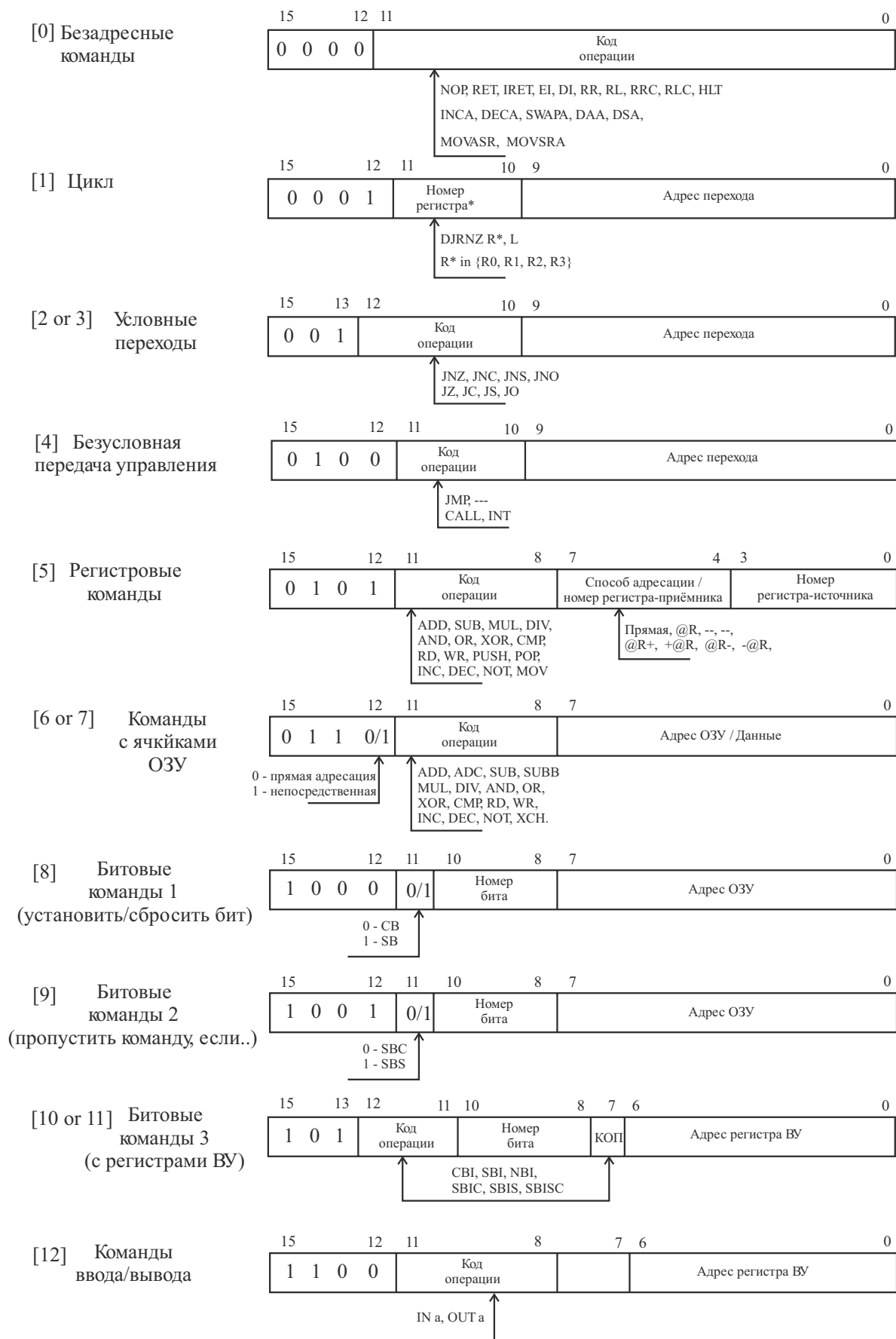


Рисунок 2. Форматы команд fN8

## 2.2. Таблица команд

В таблице приняты следующие обозначения:

Асс – содержимое аккумулятора;

DD – содержимое ячейки памяти или непосредственный операнд;

R – содержимое регистра общего назначения R;

R\* – содержимое регистра или косвенно адресуемой через регистр ячейки памяти;

p – префикс перед именем регистра, определяющий способ адресации;

A – адрес ячейки памяти данных;

M(A) – содержимое ячейки памяти данных по адресу A;

C – флаг PSW[4] переноса(заёма)

SPL – содержимое указателя стека;

L – адрес перехода (метка или абсолютный);

CR – содержимое регистра команд;

RIO – содержимое регистра внешнего устройства;

a – номер (адрес) регистра внешнего устройства;

b – номер бита в байте;

Таблица 1 — Система команд

Мнемокод	Название	Действия	Флаги				
			О	С	А	N	Z
Безадресные команды [Код 0]							
NOP	Нет операции	$PCL := PCL + 1$	-	-	-	-	-
RET	Возврат из подпрограммы	$PCL := M(SS.SPL); Inc(SPL);$ $CS := M(SS.SPL)[1:0]; Inc(SPL)$	-	-	-	-	-
IRET	Возврат из прерывания	$PCL := M(SS.SPL); Inc(SPL);$ $PSW[5:0].CS := M(SS.SPL);$ $Inc(SPL)$	-	-	-	-	-
EI	Разрешить прерывание	$FI := 1$	-	-	-	-	-
DI	Запретить прерывание	$FI := 0$	-	-	-	-	-
RR	Сдвиг аккумулятора правый циклический	$Acc[7:0] := Acc[0].Acc[7:1];$ $FC := Acc[0]$	-	-	-	-	-
RL	Сдвиг аккумулятора левый циклический	$Acc[7:0] := Acc[6:0].Acc[7];$ $FC := Acc[7]$	-	-	-	-	-
RRC	Сдвиг аккумулятора правый через перенос	$Acc[7:0] := FC.Acc[7:1];$ $FC := Acc[0]$	-	-	-	-	-
RLC	Сдвиг аккумулятора левый через перенос	$Acc[7:0] := Acc[6:0].FC;$ $FC := Acc[7]$	-	-	-	-	-
Смотри продолжение на следующей странице							

Таблица 1 — Система команд

Мнемокод	Название	Действия	Флаги				
			О	С	А	N	Z
NOTA	Инверсия аккумулятора	$Acc := \overline{Acc}$	0	0	0	+	+
INCA	Инкремент аккумулятора	$Acc := Acc + 1$	-	+	+	+	+
DECA	Декремент аккумулятора	$Acc := Acc - 1$	-	+	+	+	+
SWAPA	Обмен тетрадами аккумулятора	$Acc[7:4] \leftrightarrow Acc[3:0]$	-	-	-	-	-
DAA	Десятичная коррекция сложения	См. раздел 2.3.1	-	+	+	+	+
DSA	Десятичная коррекция вычитания	См. раздел 2.3.1	-	+	+	+	+
MOVSP	Загрузка SPL	$SPL := Acc;$	-	-	-	-	-
MOVAPSW	Прочитать PSW	$Acc := PSW$	-	-	-	-	-
MOVASR	Прочитать SR	$Acc := SR$	-	-	-	-	-
MOVSRA	Загрузить SR	$SR := Acc$	-	-	-	-	-
HLT	Стоп	Прекратить командные циклы	-	-	-	-	-
<i>Цикл [Код 1]</i>							
DJRNZ $R^c, L$	Цикл	$R^c := R^c - 1;$ if $R^c \neq 0$ then goto L	-	-	-	-	-
$R^c \in \{R0, R1, R2, R3\}$							
<i>Команды условных переходов [Код 2 or 3]</i>							
JNZ L	Переход, если не ноль	if Z=0 then CS.PCL := CR[9:0]	-	-	-	-	-
JNC L	Переход, если не перенос	if C=0 then CS.PCL := CR[9:0]	-	-	-	-	-
JNN L	Переход, если не отрицательно	if N=0 then CS.PCL := CR[9:0]	-	-	-	-	-
JNO L	Переход, если не переполнение	if O=0 then CS.PCL := CR[9:0]	-	-	-	-	-
JZ L	Переход, если ноль	if Z=1 then CS.PCL := CR[9:0]	-	-	-	-	-
JC L	Переход, если перенос	if C=1 then CS.PCL := CR[9:0]	-	-	-	-	-
JN L	Переход, если отрицательно	if N=1 then CS.PCL := CR[9:0]	-	-	-	-	-
JO L	Переход, если переполнение	if O=1 then CS.PCL := CR[9:0]	-	-	-	-	-
Смотри продолжение на следующей странице							

Таблица 1 — Система команд

Мнемокод	Название	Действия	Флаги				
			О	С	А	N	Z
Команды безусловной передачи управления [Код 4]							
JMP L	Безусловный переход	CS.PCL := CR[9:0]	-	-	-	-	-
CALL L	Вызов подпрограммы	Dec(SPL); M(SS.SPL)[1:0] := CS; Dec(SPL); M(SS.SPL) := PCL; CS.PCL := CR[9:0]	-	-	-	-	-
INT v	Вызов прерывания	Dec(SPL); M(SS.SPL) := PSW[5:0].CS; Dec(SPL); M(SS.SPL) := PCL; PCL := M(2v); CS := M(2v+1)[1:0]; FI := 0	-	-	-	-	-
v = CR[2:0] – вектор прерывания							
Регистровые команды [Код 5] (Для команд ADC и SUBB – [Код F])							
ADD pR	Сложение	Acc := Acc + R*	+	+	+	+	+
ADC pR	Сложение с переносом	Acc := Acc + R* + C	+	+	+	+	+
SUB pR	Вычитание	Acc := Acc – R*	+	+	+	+	+
SUBB pR	Вычитание с заёмом	Acc := Acc – R* – C	+	+	+	+	+
MUL pR	Умножение	R1.R0 := Acc × R*	-	-	-	-	-
DIV pR	Деление	Acc := Acc : R*	-	-	-	-	+
AND pR	Конъюнкция	Acc := Acc & R*	0	0	0	+	+
OR pR	Дизъюнкция	Acc := Acc ∨ R*	0	0	0	+	+
XOR pR	Неравнозначность	Acc := Acc ⊕ R*	0	0	0	+	+
CMP pR	Сравнение	R* – Acc	+	+	+	+	+
RD pR	Чтение	Acc := R*	-	-	-	-	-
WR pR	Запись	R* := Acc	-	-	-	-	-
XCH pR	Обмен	R* ⇔ Acc	-	-	-	-	-
PUSH R	Поместить в стек	Dec(SPL); M(SPL) := R;	-	-	-	-	-
POP R	Извлечь из стека	R := M(SPL); Inc(SPL)	-	-	-	-	-
INC R	Инкремент	R := R + 1;	-	+	+	+	+
DEC R	Декремент	R := R – 1;	-	+	+	+	+
NOT R	Инверсия	R := $\overline{R}$ ;	0	0	0	+	+
MOV R <sub>r</sub> , R <sub>t</sub>	Копирование	R <sub>r</sub> := R <sub>t</sub>	-	-	-	-	-
Смотри продолжение на следующей странице							

Таблица 1 — Система команд

Мнемокод	Название	Действия	Флаги O C A N Z
<i>Команды с ячейками ОЗУ [Код 6 or 7]</i>			
Код 6 – прямая адресация (A), 7 – непосредственная (#A)			
ADD A	Сложение	$Acc := Acc + DD$	+ + + + +
ADC A	Сложение с переносом	$Acc := Acc + DD + FC$	+ + + + +
SUB A	Вычитание	$Acc := Acc - DD$	+ + + + +
SUBB A	Вычитание с заёмом	$Acc := Acc - DD - FC$	+ + + + +
MUL A	Умножение	$R1.R0 := Acc \times DD$	- - - - -
DIV A	Деление	$Acc := Acc : DD$	- - - - +
AND A	Конъюнкция	$Acc := Acc \& DD$	0 0 0 + +
OR A	Дизъюнкция	$Acc := Acc \vee DD$	0 0 0 + +
XOR A	Неравнозначность	$Acc := Acc \oplus DD$	0 0 0 + +
CMP A	Сравнение	$DD - Acc$	+ + + + +
RD A	Чтение	$Acc := DD$	- - - - -
WR A	Запись	$M(A) := Acc$	- - - - -
XCH A	Обмен	$M(A) \Leftrightarrow Acc$	- - - - -
INC A	Инкремент	$M(A) := M(A) + 1$	- + + + +
DEC A	Декремент	$M(A) := M(A) - 1$	- + + + +
NOT A	Инверсия	$M(A) := \overline{DD}$	0 0 0 + +
<i>Битовые команды 1 [Код 8]</i>			
CB A,b	Сбросить бит	$M(A)[b] := 0$	- - - - -
SB A,b	Установить бит	$M(A)[b] := 1$	- - - - -
<i>Битовые команды 2 [Код 9]</i>			
SBC A,b	Пропустить команду, если бит сброшен	if $M(A)[b]=0$ then $PCL := PCL + 1$	- - - - -
SBS A,b	Пропустить команду, если бит установлен	if $M(A)[b]=1$ then $PCL := PCL + 1$	- - - - -
<i>Битовые команды 3 [Код A or B]</i>			
CBI a,b	Сбросить бит	$RIO(a)[b] := 0$	- - - - -
SBI a,b	Установить бит	$RIO(a)[b] := 1$	- - - - -
NBI a,b	Инвертировать бит	$RIO(a)[b] := \overline{RIO(a)[b]}$	- - - - -
SBIC a,b	Пропустить команду, если бит сброшен	if $RIO(a)[b]=0$ then $PCL := PCL + 1$	- - - - -
SBIS a,b	Пропустить команду, если бит установлен	if $RIO(a)[b]=1$ then $PCL := PCL + 1$	- - - - -
Смотри продолжение на следующей странице			

Таблица 1 — Система команд

Мнемокод	Название	Действия	Флаги				
			О	С	А	N	Z
SBISC a,b	Пропустить команду, если бит установлен и сбросит бит	if RIO(a)[b]=1 then (PCL := PCL + 1; RIO(a)[b] := 0)	-	-	-	-	-
<i>Команды ввода/вывода [Код C]</i>							
IN a	Ввод	Acc := RIO(a)	-	-	-	-	-
OUT a	Вывод	RIO(a) := Acc	-	-	-	-	-

## 2.3. Пояснения к некоторым командам таблицы 1

### 2.3.1. Безадресные команды десятичной коррекции

**DAA** – десятичная коррекция байта в Асс после сложении (кодировка «8421»)

$$\text{if } (B_L > 9) \text{ or } (AC) \text{ then } B := B + 0x06; \quad (1)$$

$$\text{if } (B_H > 9) \text{ or } (C) \text{ then } B := B + 0x60; \quad (2)$$

**DSA** – десятичная коррекция байта в Асс после вычитании (кодировка «8421»)

$$\text{if } (AC) \text{ then } B := B - 0x06; \quad (3)$$

$$\text{if } (C) \text{ then } B := B - 0x60; \quad (4)$$

Здесь  $B$  – предварительный результат операции сложения (вычитания),  $B_H$  – старшая тетрада  $B$ ,  $B_L$  – младшая тетрада  $B$ ,  $AC$  – перенос (заём) из младшей тетрады,  $C$  – перенос (заём) из байта.

### 2.3.2. Умножение

Содержимое аккумулятора можно умножить на содержимое регистра, прямо или косвенно адресуемой ячейки памяти или 8-разрядную константу. Формат произведения – два байта, поэтому оно размещается не в аккумуляторе, а в регистрах R1 и R0, причём в R1 размещается старший байт произведения, а в R0 – младший.

### 2.3.3. Деление

Целочисленное деление: байт Асс делится на байт делителя, целая часть частного помещается в Асс. Если содержимое Асс меньше делителя, то результат деления равен 0, устанавливается флаг  $Z = 1$ . В случае ненулевого результата деления флаг  $Z = 0$ , остальные флаги в операции деления не изменяются. В качестве делителя можно использовать непосредственный операнд, содержимое регистра, содержимое прямо или косвенно адресуемой ячейки памяти.



#### 2.3.4. Цикл

**DJRNZ R,L** – декремент регистра и проверка. Если после декремента содержимое регистра  $\neq 0$ , то осуществляется переход по указанному адресу, иначе – на следующую команду. Можно использовать для организации циклов. Работает только с регистрами R0, R1, R2, R3.

#### 2.3.5. Команды безусловной передачи управления

**CALL L** – вызов подпрограммы по прямому адресу. Использует две ячейки стека для размещения 10-разрядного адреса возврата (в старшем байте занято только два младших разряда).

**INT v** – вызов обработчика прерывания по вектору v. Как и команда CALL, сохраняет адрес возврата, а в свободные 6 разрядов старшего байта помещает флаги I, O, C, AC, N, Z. Таблица векторов прерываний в ОЗУ начинается с адреса 0x000, вектор прерываний  $v \in \{0, 1, \dots, 7\}$ . Адрес обработчика прерываний загружается из ячеек с адресами  $(2v)$  и  $(2v+1)[1:0]$  в PCL и SR[1:0] соответственно.

### 3. Внешние устройства

Модели внешних устройств (ВУ), используемые в описываемой системе, реализованы по единому принципу. С точки зрения процессора они представляют собой ряд программно-доступных регистров, лежащих в **адресном пространстве ввода/вывода**. Размер регистров ВУ совпадает с размером ячеек памяти и регистров данных процессора — один байт.

Доступ к регистрам ВУ осуществляется по командам *IN a*, *OUT a*, где *a* — семи-разрядный двоичный адрес регистра ВУ. Таким образом, общий объем адресного пространства ввода/вывода составляет  $2^7 = 128$  адресов. Следует помнить, что адресные пространства памяти и ввода/вывода в этой модели разделены.

Разные ВУ содержат различное число программно-доступных регистров, каждому из которых соответствует свой адрес, причем нумерация адресов в каждом ВУ начинается с 0. При подключении ВУ ему средствами *Диспетчера внешних устройств* ставится в соответствие *базовый адрес* в пространстве ввода/вывода и все адреса регистров ВУ становятся *смещениями* относительно его базового адреса.

#### 3.1. Диспетчер внешних устройств

Окно диспетчера (рис. 3) отображает список ВУ, доступных для подключения в систему. Каждому ВУ предлагается базовый адрес по умолчанию, а тем устройствам, которые работают с прерываниями – и вектор прерывания по умолчанию.

Пользователь может произвольно изменить предлагаемые по умолчанию параметры. При этом надо учитывать, что базовый адрес ВУ в пространстве ввода/вывода всегда должен быть кратным 16, поэтому в окне диспетчера можно задать только старшую цифру шестнадцатеричного базового адреса в диапазоне от 0 до 7. При выборе вектора

прерывания следует учитывать, что вектор 0 зарезервирован за сигналом RESET, а остальные вектора от 1 до 7 могут назначаться ВУ произвольно.

Не следует назначать разным устройствам одинаковые базовые адреса и вектора прерывания!

Подключение ВУ осуществляется нажатием кнопки с соответствующим именем в окне диспетчера. Отключение подключённых ВУ в процессе работы fN8 не предусмотрено.

Возможна работа с несколькими одинаковыми ВУ, если при последовательном подключении им будут назначены разные базовые адреса и вектора прерывания.

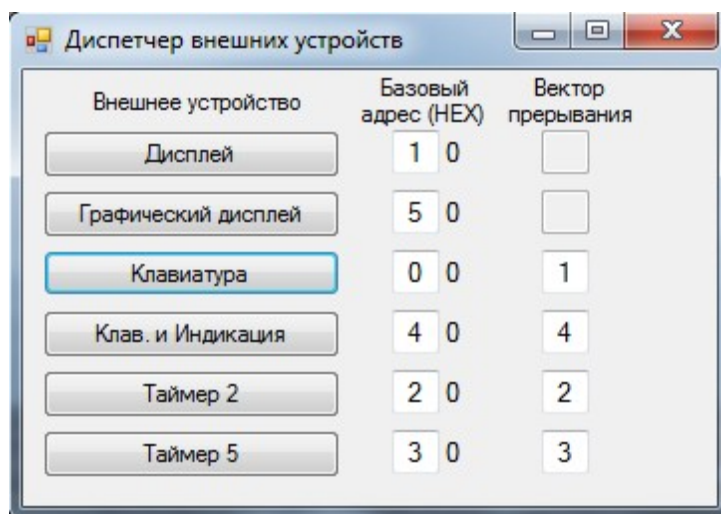


Рисунок 3. Окно диспетчера ВУ

### 3.2. Контроллер клавиатуры

Контроллер клавиатуры является двоичным функциональным аналогом контроллера клавиатуры, входящего в состав модели десятичной ЭВМ *CompModel* [1, 2]. Контроллер содержит 64-символьный буфер ASCII-кодов нажатых клавиш, три программно-доступных регистра, две кнопки и схему управления (рис. 4).

В состав контроллера клавиатуры входят три программно-доступных регистра:

- DR (адрес 0) — регистр данных;
- CR (адрес 1) — регистр управления, определяет режимы работы контроллера и содержит следующие флаги:

*E* — флаг разрешения приема кодов в буфер;

*I* — флаг разрешения формирования запроса на прерывание;

*S* — флаг режима ввода.

- SR (адрес 2) — регистр состояния, доступен по чтению и записи, содержит два флага:

*Err* — флаг ошибки;

*Rdy* — флаг готовности.

**Регистр данных DR** доступен только для чтения, через него считываются ASCII-коды из буфера, причем порядок чтения кодов из буфера соответствует порядку их записи в буфер — каждое чтение по адресу 0 автоматически перемещает указатель чтения буфера. В каждый момент времени DR содержит код символа по адресу указателя чтения буфера.

Флаги **регистра управления CR** устанавливаются и сбрасываются программно.

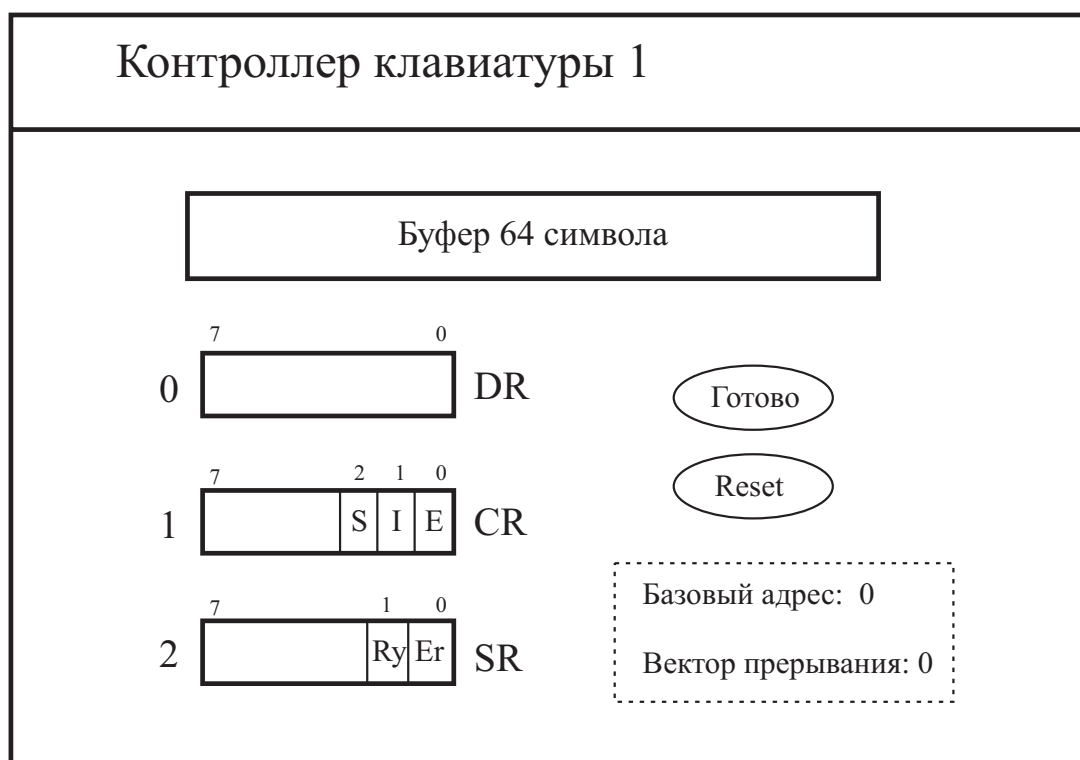


Рисунок 4. Контроллер клавиатуры

Флаг  $E$ , будучи установленным, разрешает приём кодов в буфер. При  $E = 0$  контроллер игнорирует нажатие на клавиатуре, приём кодов в буфер прекращается. На считывание кодов из буфера флаг  $E$  влияния не оказывает.

Флаг  $I$ , будучи установленным, разрешает при определенных условиях формирование контроллером запроса на прерывание. При  $I = 0$  запрос на прерывание не формируется.

Состояние флага  $S$  определяет условие формирования флага готовности Rdy (Ready) в регистре SR. В режиме *посимвольного ввода* ( $S = 0$ , по умолчанию) флаг Rdy устанавливается после ввода в буфер кода каждого символа, а сбрасывается аппаратно после считывания кода символа из регистра данных DR.

В режиме *строчного ввода* ( $S = 1$ ) флаг Rdy устанавливается только после нажатия в окне обозревателя клавиатуры кнопки *Готово*; сбрасывается флаг Rdy так же, как в режиме посимвольного ввода.

Запрос на прерывание всегда формируется по условию  $(I = 1) \& (Rdy = 1)$ .

Флаг ошибки Err (Error) в регистре SR устанавливается при попытке ввода в буфер 65-го символа. Ввод 65-го и всех последующих символов блокируется.

Сброс флага Rdy осуществляется автоматически при чтении из регистра DR, флаг Err сбрасывается программно. Кроме того, оба эти флага сбрасываются при нажатии кнопки *Сброс* на панели обозревателя; одновременно со сбросом флагов производится очистка буфера – весь буфер заполняется кодами 00h и указатели записи и чтения устанавливаются в начало буфера.

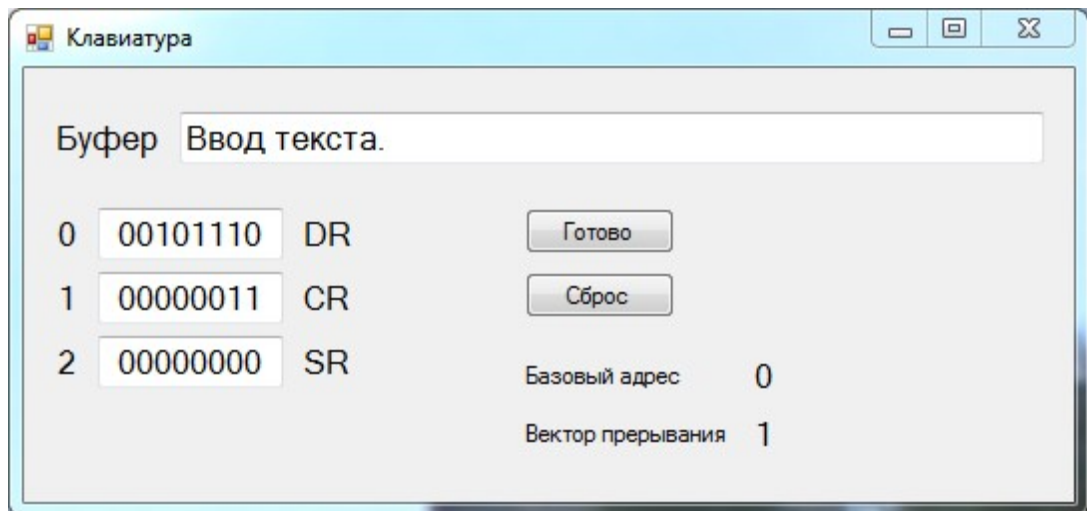


Рисунок 5. Окно обозревателя «Контроллер клавиатуры»

### 3.3. Символьный дисплей

Дисплей может отображать символы, задаваемые ASCII-кодами, поступающими на его регистр данных. Окно обозревателя дисплея показано на рис. 6.

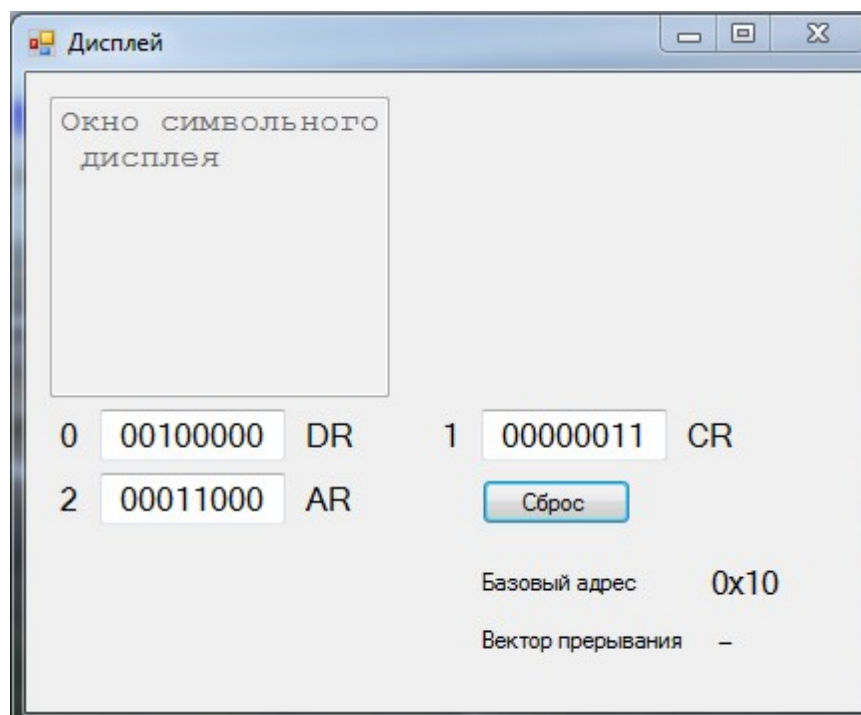


Рисунок 6. Окно обозревателя «Дисплей»

Дисплей включает:

- видеопамять объемом 128 байт (ОЗУ дисплея);
- символьный экран размером 8 строк по 16 символов в строке;
- три программно-доступных регистра:

DR (адрес 0) — регистр данных;  
CR (адрес 1) — регистр управления;  
AR (адрес 2) — регистр адреса;

Через регистры **адреса AR** и **данных DR**, доступные по записи и чтению, осуществляется доступ к ячейкам видеопамати. При обращении к регистру DR по записи содержимое аккумулятора записывается в DR и в ячейку видеопамати, адрес которой установлен в регистре AR. При чтении DR в аккумулятор загружается содержимое ячейки видеопамати по адресу AR.

Регистр **управления CR** доступен только по записи и содержит два флага:

$CR[0] = E$  — флаг разрешения работы дисплея; при  $E = 0$  запись в регистры AR и DR блокируется.

$CR[1] = A$  — флаг автоинкремента адреса; при  $A = 1$  содержимое AR автоматически увеличивается на 1 после любого обращения к регистру DR — по записи или чтению.  $CR[7 : 2]$  — не используются

### 3.4. Таймер-2

В состав набора ВУ, которые могут подключаться к fN8 входит несколько таймерных устройств с различными функциональными возможностями. Наиболее простым из них можно считать Таймер-2, функциональная схема которого показана на рис. 7.

Таймер-2 включает:

- 16-разрядный суммирующий счётчик TCNT с программируемым предделителем,
- 16-разрядный регистр TIOR захвата/автозагрузки,
- схему сравнения  $TCNT == TIOR$ ,
- 16-разрядный регистр TSCR управления/состояния,
- буферный регистр старшего байта Buf\_H.

#### 3.4.1. Режимы работы Таймера-2

Таймер-2 может работать в одном из четырёх режимов:

Режим 0 — простой счёт. В этом режиме состояние TCNT инкрементируется с частотой, определяемой коэффициентом деления предделителя. При переходе счётчика из состояния 0xFFFF в состояние 0x0000 устанавливается флаг переполнения OVF, формируется запрос на прерывание, если установлен флаг прерывания по переполнению OVIE, счёт продолжается.

Режим 1 — режим захвата. Здесь по внешнему (по отношению к Таймеру-2) сигналу текущее состояние TCNT копируется в TIOR, устанавливается флаг захвата ICF и формируется запрос на прерывание при условии ICIE = 1, счёт продолжается. Источником сигнала «Захват» может служить одноимённая кнопка в окне обозревателя Таймера-2 (рис. 8) или установленный программно бит IC в регистре TSCR.

Режим 2 – сравнение. В этом режиме предварительно устанавливается произвольное значение в регистр TIOR и с ним сравнивается текущее значение TCNT. При  $TCNT = TIOR$  устанавливается флаг сравнения OCF и формируется запрос на прерывание при условии OCIE = 1. Если разряд RT регистра TSCR сброшен, то счёт продолжается далее, а если  $RT = 1$ , то TCNT обнуляется и счёт начинается с начала. Фактически такой режим аналогичен простому счёту, верхний предел которого определяется содержимым TIOR.

Режим 3 – автозагрузка. Этот режим позволяет начинать счёт не с 0, а со значения, установленного в TIOR. Счёт идёт до значения 0xFFFF, устанавливается флаг переполнения OVF, может быть сформирован запрос на прерывание, а в TCNT копируется значение из TIOR.

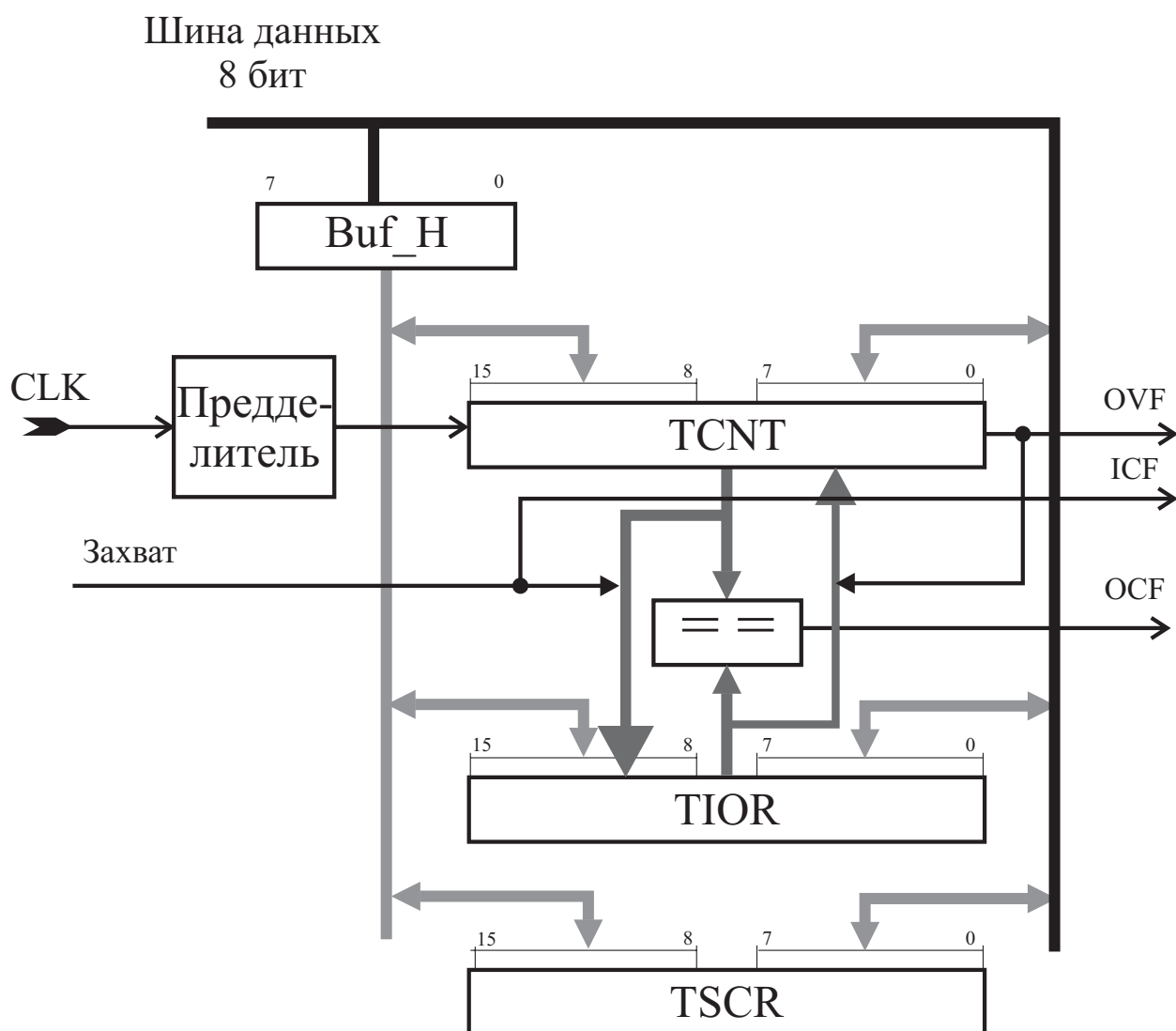


Рисунок 7. Функциональная схема Таймера-2

### 3.4.2. Формат и назначение разрядов регистра TSCR

№ бита	Имя	Назначение
0	T	Включить таймер
1..2	Code	Режим
3	RT	Сброс TCNT при сравнении
4	OVIE	Разрешить прерывание при переполнении
5	OCIE	Разрешить прерывание при сравнении
6	ICIE	Разрешить прерывание при захвате
7	0	Разрешение загрузки TSCRL
8(0)	IC	Программный «Захват»
9(1)	OVF	Флаг переполнения
10(2)	OCF	Флаг сравнения
11(3)	ICF	Флаг захвата
14(6)..12(4)	k	Коэффициент деления предделителя
15(7)	0	Разрешение загрузки TSCRH

Флаги OVF, OCF, и ICF устанавливаются аппаратно при возникновении соответствующего события и сбрасываются только программно.

Разряд IC – «Захват» может устанавливаться программно или по нажатию кнопки *Захват* в окне обозревателя *Таймера 2*. Установка бита IC в режиме «01» немедленно вызывает захват содержимого TCNT в TIOR, установку флага ICF и при ICIE = 1 – формирование запроса на прерывание. Кроме того, после установки ICF бит IC автоматически сбрасывается. В других режимах работы таймера установка IC игнорируется.

### 3.4.3. Особенности организации доступа к 16-разрядным регистрам по 8-разрядному интерфейсу

Очевидно, что в адресном пространстве 8-разрядной ЭВМ 16-разрядный регистр должен занимать два соседних адреса, причём обычно младший байт получает чётный адрес, а старший – нечётный. Выполнив пару команд обращения, можно прочитать или загрузить 16-разрядный код. Проблемы могут возникнуть, если состояние 16-разрядного регистра может изменяться с достаточно высокой частотой. Так, при считывании, например, значения TCNT = 0x03FF переходящее в 0x0400, один байт может быть прочитан из «старого» кода, а другой – из «нового» (0x0300 или 0x04FF) – ошибка значительно превысит величину младшего разряда!

Чтобы исключить подобные ситуации в таймерных системах принято осуществлять буферирование старшего байта 16-разрядных слов. Чтение<sup>3</sup> двухбайтового слова следует всегда начинать с младшего байта, при этом старший байт слова одновременно копируется в буферный регистр. При чтении старшего байта слова, его значение извлекается не из регистра, а из буфера. Таким образом, оба считанных байта слова относятся к одному моменту дискретного времени.

Аналогично при записи (см. сноску 3) в двухбайтовое слово сначала следует записывать по старшему адресу, при этом байт записывается в буфер, а старший байт 16-разрядного регистра не меняется. По команде записи в младший байт слова одно-

<sup>3</sup> Не забываем, что чтение из регистров ВУ осуществляется командами *IN a*, а запись – *OUT a*.



временно производится запись в старший байт из буфера, таким образом оба байта слова попадают в 16-разрядный регистр одновременно.

Единственный буфер Buf\_H в Таймере-2 является общим для всех трёх 16-разрядных регистров, поэтому, если требуется записать в регистры константы с одинаковыми старшими байтами, достаточно один раз записать константу в буфер.

Следует помнить, что в операции записи байта в TSCRH значение разрядов  $b_3 \dots b_1$  безразлично. Эти разряды соответствуют флагам OVF, OCF, и ICF, которые устанавливаются только аппаратно, а при записи любого байта в TSCRH сохраняют свои прежние значения.

Сбросить эти флаги можно с помощью команд записи бита.

В регистре управления/состояния TSCR часто требуется изменить отдельные биты, поэтому в механизме загрузки TSCR предусмотрены дополнительные возможности.

Если необходимо произвести запись байтов в TSCR (старшего, потом младшего) следует соблюдать дополнительное условие: старшие биты  $b_7$  этих байтов должны быть «0». Если по адресу TSCRH (0x25) или TSCRL (0x24) производится запись байта с  $b_7 = 1$ , то это интерпретируется Таймером-2 как команда записи бита в адресуемый регистр, причём разряды  $b_3 \dots b_1$  определяют номер бита, а  $b_0$  – устанавливаемое значение<sup>4</sup>.

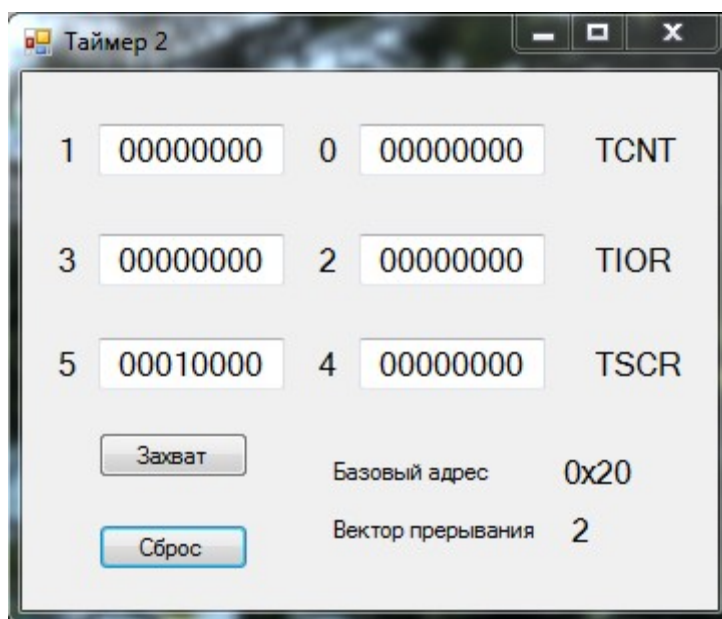


Рисунок 8. Окно обозревателя Таймер-2

### 3.5. Таймер-5

Функциональная схема модуля Таймера-5 показана на рисунке 9, а окно обозревателя – на рисунке 10.

Модуль включает:

- 16-разрядный реверсивный счётчик TCNT (относительные адреса 1, 0),
- делитель частоты счёта,
- 16-разрядный регистр сравнения OCR (3, 2),
- схему сравнения  $CNT = IOR$ ,
- 16-разрядный регистр захвата ICR (5, 4),
- 8-разрядный буферный регистр старших байтов BUFH,
- два 8-разрядных регистра состояния/управления TSCRL (6) и TSCRH (7),
- блок управления таймером,

<sup>4</sup>В fN8 можно воспользоваться битовыми командами *CBI a,b* или *SBI a,b*.



- формирователь выходного сигнала.

При подключении к fN8 Таймеру-5 по умолчанию присваивается базовый адрес 0x30<sup>5</sup>, адресное пространство таймера составляет 8 байт (адреса 0x37..0x30).

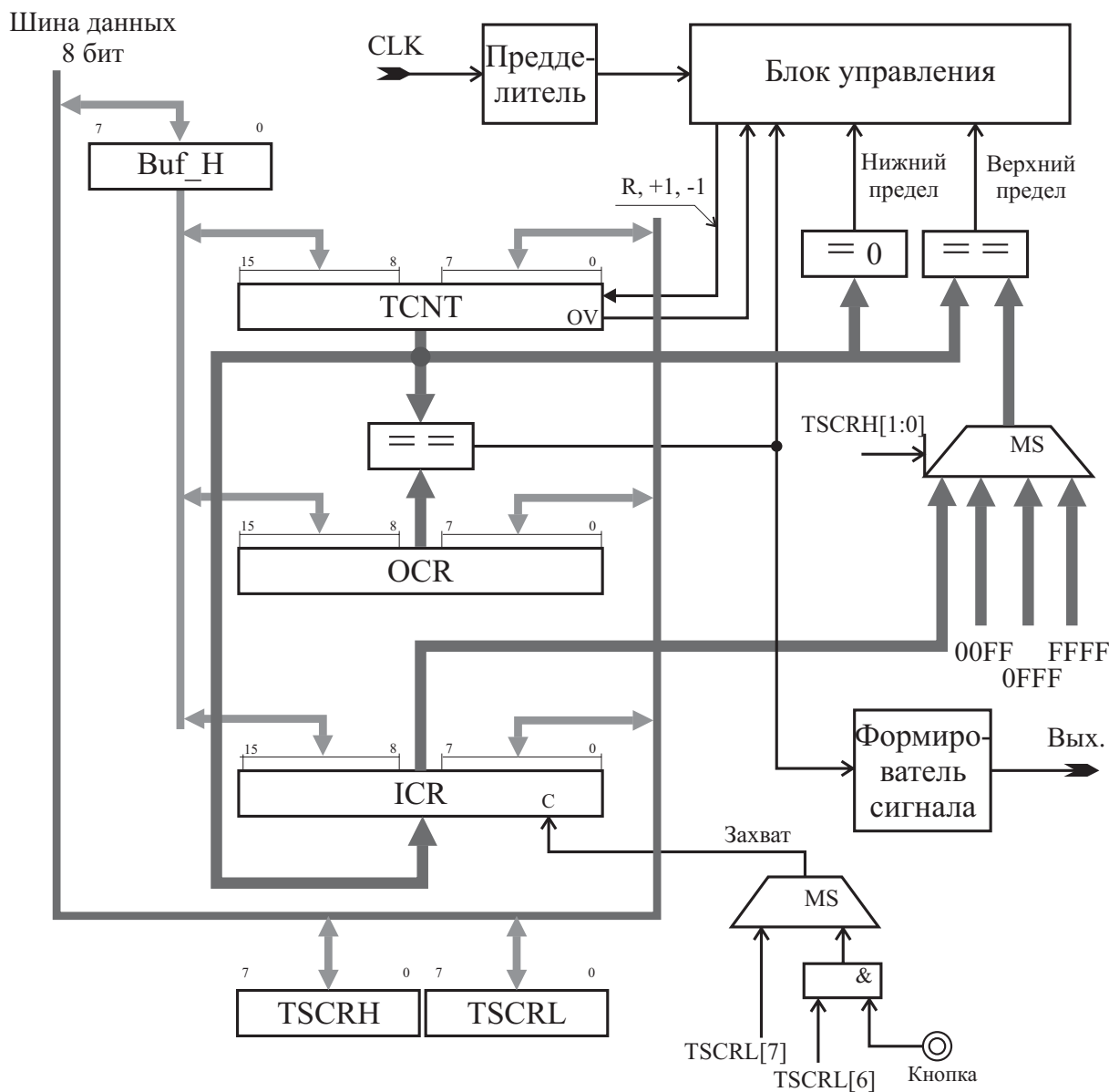


Рисунок 9. Функциональная схема Таймера-5

Таймер-5 может формировать запросы на прерывания по трём различным событиям:

- превышение заданного верхнего предела счёта,
- захват состояния TCNT в ICR
- совпадение значений TCNT и IOR.

Все три события генерируют запрос на прерывание с вектором 3 (по умолчанию,

<sup>5</sup>Напомним, что при необходимости базовый адрес при подключении может быть изменён (см. раздел 3.1).

можно изменить), а идентификация конкретного события должна выполняться программно в обработчике прерывания путём анализа соответствующих флагов регистра TSCRH.

Верхний предел счёта устанавливается программно (поле TSCRH[1:0]) и может принимать значение одной из трёх констант: 0x00FF, 0x0FFF, 0xFFFF. Кроме того, в качестве значения верхнего предела может выступать содержимое регистра ICR, в этом случае в качестве верхнего предела может быть выбрано любое значение от 0 до 0xFFFF.

### 3.5.1. Режимы работы

Модуль поддерживает следующие режимы работы:

Режим **0** – **Просто счёт**: инкремент TCNT, формирование флага OVF при переходе *Верхний предел*  $\rightarrow$  0.

Режим **1** – **Сброс TCNT при совпадении** TCNT = OCR, формирование флага ICF.

Режим **2** – **Быстрый ШИМ**;

Режим **3** – **ШИМ с фазовой коррекцией** (ШИМ ФК).

### 3.5.2. Форматы регистров состояния/управления

	7	6	5	4	3	2	1	0
TSCRL (0x36)	ПЗ	РВЗ	Внешний сигнал	Частота	Режим			

Назначение полей регистра TSCRL:

TSCRL[1:0] – определяет режим работы (0..3);

TSCRL[3:2] – задаёт частоту счёта (управление предделителем):

00 – счётчик выключен,

01 – OSC (частота задающего генератора),

10 – OSC/16,

11 – OSC/64;

TSCRL[5:4] – управление формированием внешнего сигнала (зависит от выбранного режима, см. табл. в разделе 3.5.3);

TSCRL[6] – разрешение внешнего захвата;

TSCRL[7] – программный захват.

	7	6	5	4	3	2	1	0
TSCRH (0x37)	ICF	OCF	OVF	ICIE	OCIE	OVIE	Верхний предел	

Назначение полей регистра TSCRH:

TSCRH[1:0] – определяет значение верхнего предела:

00 – ICR

01 – 00FF  
 10 – 0FFF  
 11 – FFFF

TSCRH[4:2] – биты *разрешения прерываний* по событиям: внешний захват (ICIE), совпадение TCNT = OCR (OCIE), таймер достиг верхнего предела (OVIE);  
 TSCRH[7:5] – флаги соответствующих событий: ICF, OCF, OVF.

### 3.5.3. Формирование выходного сигнала

TSCRL[5:4]	Без ШИМ	Быстрый ШИМ	ШИМ ФК
00	Выход отключён		
01	Инверсия при совпадении TCNT = OCR		
10	Сброс («0») с при совпадении	Сброс при совпадении, установка на верхнем пределе счёта	Сброс при совпадении во время прямого счёта, установка при совпадении во время обратного счёта
11	Установка («1») при совпадении	Установка при совпадении, сброс на верхнем пределе счёта	Установка при совпадении во время прямого счёта, сброс при совпадении во время обратного счёта

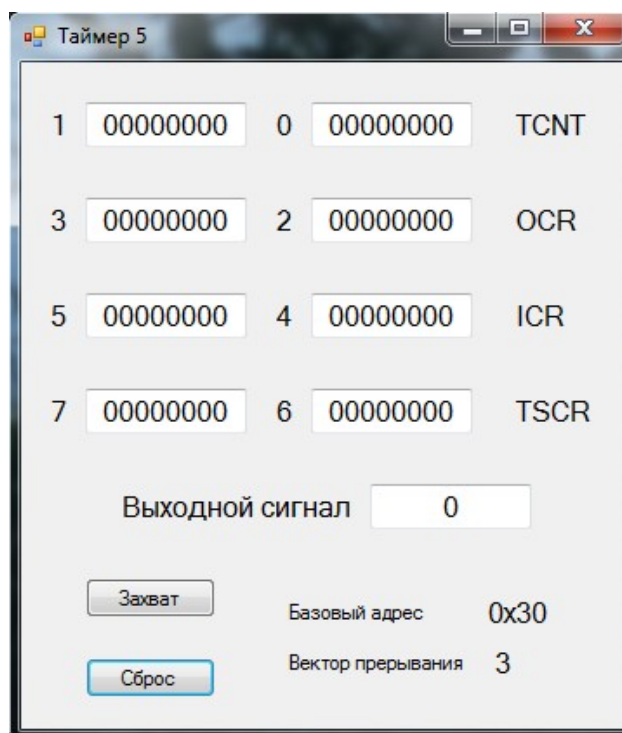


Рисунок 10. Окно обозревателя Таймера-5

### 3.5.4. Буферирование старшего байта

Для того, чтобы извлекаемые/записываемые данные 16-разрядных регистров, передаваемые по 8-разрядной шине, *относились к одному моменту времени* используется буферирование старшего байта регистров.

При чтении из регистра программа должна обращаться сначала к его младшему байту. При этом содержимое старшего байта в том же такте автоматически копируется в буфер Buf\_H. Команда чтения из старшего байта регистра считывает информацию из Buf\_H. Таким образом, оба считанных из регистра байта относятся к одному моменту времени.

При записи следует сначала записывать старший байт, при этом он всегда попадает в Buf\_H. По команде записи в регистр младшего байта старший байт одновременно загружается в регистр из Buf\_H.

Заметим, что обращение в регистр состояния/управления TSCR не буферизируется – обращение выполняется непосредственно в TSCRH, причём обращаться к TSCRH и TSCRL можно в любом порядке.

### 3.5.5. Описание режимов работы

**Режим 0 – Простой счёт.** TCNT инкрементируется от нижнего предела (всегда 0) до определённого в поле TSCRH[1:0] верхнего предела счёта<sup>6</sup> с частотой, определяемой полем TSCRL[3:2]. В момент достижения верхнего предела TCNT сбрасывается в 0x0; устанавливается флаг переполнения OVF; формируется запрос на прерывание с вектором 3, если OVIE = 1; продолжается счёт.

В Режиме 0 можно использовать **функцию захвата** состояния TCNT в регистр ICR. Захват выполняется при нажатии кнопки *Захват*, если установлено разрешение внешнего захвата (TSCRL[6] = 1) или безусловно – по установке флага *Программный захват* (TSCRL[7]). При внешнем захвате устанавливается флаг ICF и может быть сформирован запрос на прерывание с вектором 3, если ICIE = 1.

**Примечание.** Отдельно следует сказать о захвате в ситуации, когда верхний предел задаётся содержимым регистра ICR. Внешний захват (если TSCRL[6] = 1) выполняется успешно, при этом заданное в ICR значение верхнего предела сохраняется во внутреннем регистре и продолжает определять верхний предел счёта. Программный захват выполняется, при этом значение верхнего предела становится неопределённым.

**Режим 1 – Сброс по совпадению.** Так же, как и в Режиме 0, выбирается частота счёта, верхний предел и, кроме того, загружается константа в регистр OCR. При совпадении TCNT = OCR устанавливается флаг совпадения OCF; при установленном флаге OCIE формируется запрос на прерывание; TCNT сбрасывается в 0x0 и продолжает счёт; возможно изменение выходного сигнала в соответствии с установками поля TSCRL[5:4] (см. табл. в разделе 3.5.3).

Очевидно, значение OCR должно быть меньше верхнего предела счёта, иначе совпадение TCNT = OCR не наступит никогда.

В режиме 1 функция захвата работает как по внешнему, так и по программному сигналу.

---

<sup>6</sup>При TSCRH[1:0] = 00 значение верхнего предела программно задаётся в регистр ICR.

**Режим 2 – Быстрая ШИМ.** В этом режиме период импульса ШИМ определяется значением верхнего предела (и, разумеется, выбранной тактовой частотой), а длительность импульса – значением регистра OCR. По достижении верхнего предела TCNT сбрасывается в 0 и начинается новый период ШИМ (рис. 11).

Значение регистра OCR должно выбираться всегда *меньше* установленного верхнего предела счёта (ВПС). Тогда в каждом периоде ШИМ можно выделить два события:  $TCNT = ВПС$  и  $TCNT = OCR$ . Поле  $(TSCRL[5:4])$  определяет, по какому из этих двух событий выходной сигнал устанавливается в «1», а по какому – сбрасывается в «0».

В режиме 2 не формируется флаг совпадения OCF, но при достижении верхнего предела счёта устанавливается OVF, что позволяет при необходимости «поймать» момент завершения очередного периода ШИМ.

На фоне генерации ШИМ-импульсов можно захватывать текущее состояние TCNT в регистр ICR программно (команда `sbi 0x36,7`) или по внешнему сигналу (кнопка *Захват* обозревателя), причём внешний захват возможен при условии  $TSCRL[6] = 1$ . Относительно захвата в случае, когда для задания верхнего предела счёта используется регистр ICR см. Примечание на стр. 28.

**Режим 3 – ШИМ с фазовой коррекцией.** Для некоторых применений ШИМ важным является расположение импульса в периоде ШИМ. Режим 3 обеспечивает выдачу импульса, симметричного относительно середины периода<sup>7</sup> ШИМ при любой длительности (рис. 12).

Период ШИМ ФК определяется временем перехода  $TCNT\ 0 \rightarrow ВПС \rightarrow 0$ , то есть при прочих равных условиях он вдвое длиннее периода быстрого ШИМ. Изменение значения выходного сигнала происходит при совпадении  $TCNT = OCR$  на участках прямого и обратного счёта TCNT.

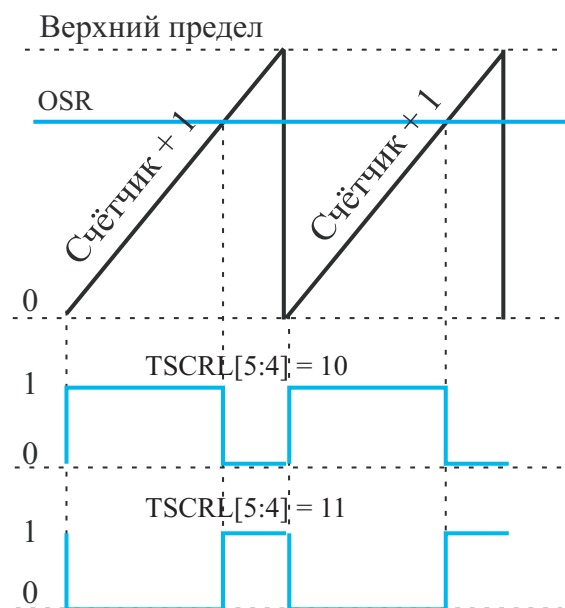


Рисунок 11. Быстрая ШИМ

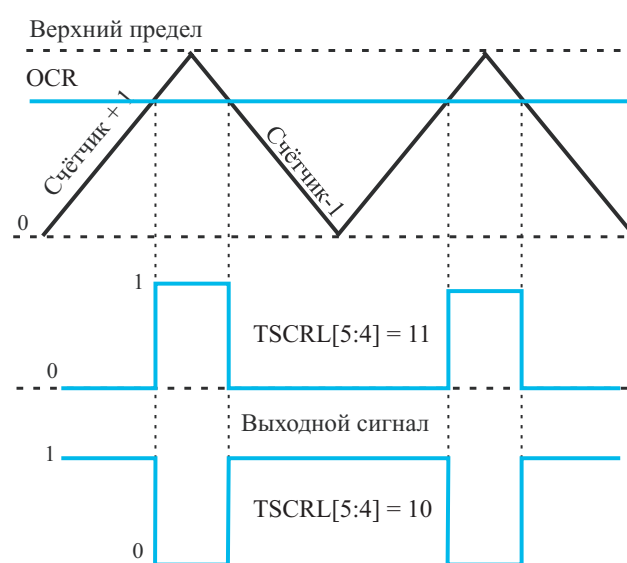


Рисунок 12. ШИМ с фазовой коррекцией

<sup>7</sup>Инверсный выходной сигнал симметричен относительно начала периода (см. рис. 12 при  $TSCRL[5:4] = 10$ ).

### 3.6. Контроллер матричной клавиатуры и 7-сегментной динамической индикации

Контроллер предназначен для управления динамической 7-сегментной индикацией и матричной клавиатурой.

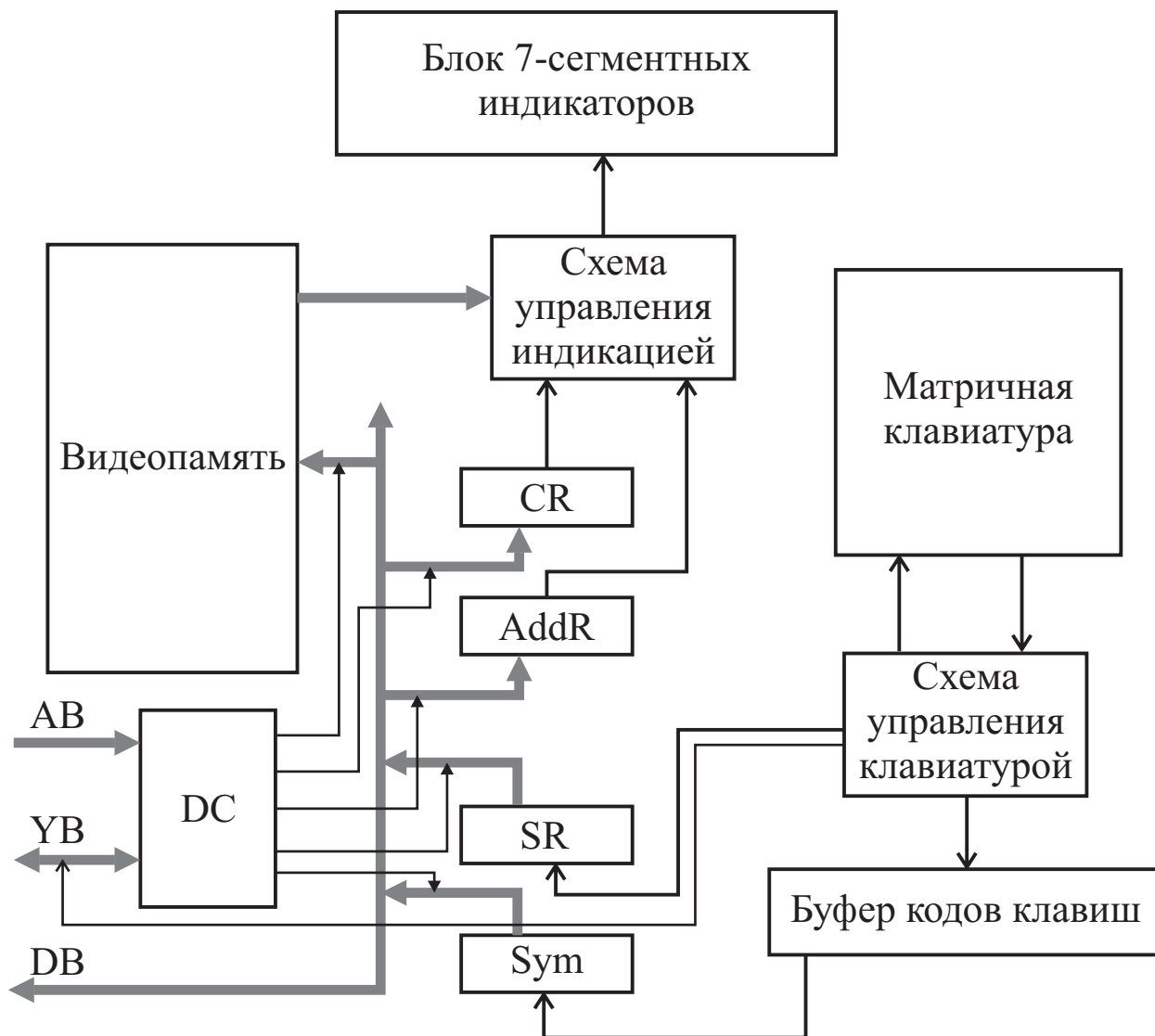


Рисунок 13. Функциональная схема контроллера клавиатуры и индикации

Функциональная схема контроллера представлена на рис. 13. Он состоит из двух слабо связанных между собой частей:

- 1) схем, управляющих выводом символов на блок 7-сегментных индикаторов:
  - видеопамять, хранящая 7-сегментные коды отображаемых символов;
  - программно-доступный регистр адреса видеопамети Addr;
  - программно-доступный регистр управления CR;
  - логических схем управления индикацией;
- 2) схем, управляющих сканированием клавиш, определением кода нажатой клавиши, сохранением кодов в буфере и информированием процессора о событии нажатия клавиши:
  - логические схемы управления модулем клавиатуры;
  - буфер кодов нажатых клавиш («очередь»);
  - программно-доступный регистр состояния контроллера SR;
  - программно-доступный регистр Sum кода клавиши (вершина буфера).

Контроллер связан с процессором по системной магистрали, включающей шину данных *DB* (8 бит), шину адреса *AB* (7 бит)<sup>8</sup> и линии шины управления *YB*: *RDIO* – чтение из регистра ввода/вывода, *WRIO* – запись в регистр ввода/вывода.

К контроллеру можно подключать блоки 7-сегментных индикаторов и матрицы клавиатуры различной размерности. Настройки контроллера позволяют выбрать размерность матрицы клавиатуры  $3 \times 3$ ,  $3 \times 4$ ,  $4 \times 4$  и блоки индикаторов на 2, 4, 6, или 8 разрядов.

Допускается работа контроллера только с модулем индикации, без подключения клавиатуры.

Настройки позволяют установить положение точки (сегмент Н) в формате 7-сегментного кода символа: GFEDCBAH (по умолчанию) или HGFEDCBA.

Окно обозревателя устройства в режиме «8 индикаторов и клавиатура  $4 \times 4$ » показано на рис. 15.

В адресном пространстве ввода/вывода контроллер использует пять адресов:

- 0 – регистр адреса видеопамети Addr. Значение этого регистра всегда устанавливается по модулю выбранной разрядности индикации.
- 1 – регистр кода клавиши Sum отображает двоичный код первого элемента буфера (очереди введённых символов клавиатуры). Код клавиши соответствует двоичному коду цифры, на ней изображенной, причём «\*» ~ E (1110), а «#» ~ F (1111).

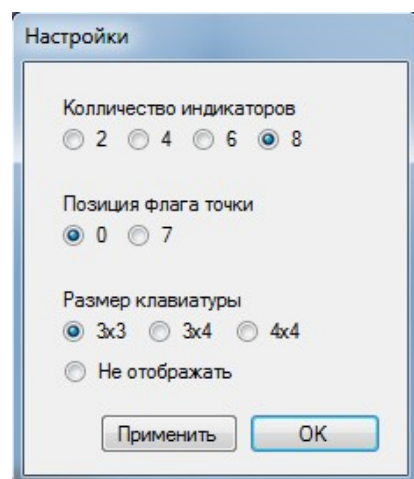


Рисунок 14. Окно настройки

<sup>8</sup>Шина адреса *AB* системной магистрали fN8 имеет 10 линий, внешние устройства используют только 7 младших из них, причём 4 младших разряда *AB*[3 : 0] используются для выбора регистра в составе ВУ, а *AB*[6 : 4] – номер ВУ (базовый адрес: *AB*[6 : 4].0000).



2 – регистр управления CR содержит три флага управления режимами работы контроллера:

CR[0] = E – флаг включения контроллера;

CR[1] = A – флаг разрешения автоинкремента Addr (по модулю выбранной разрядности индикации!) после каждой записи в видеопамять;

CR[2] = I – флаг разрешения формирования запроса на прерывание при не пустом буфере клавиатуры.

3 – регистр состояния SR содержит информацию о состоянии буфера клавиатуры:

SR[3:0] – количество находящихся в буфере не считанных кодов клавиш. Учитывая, что размер буфера равен 8, при SR[3] = 1 считается, что буфер заполнен, дальнейшая запись в него блокируется и коды нажимаемых клавиш теряются. При считывании из буфера (из регистра Sym) очередного символа значение SR[3:0] декрементируется и запись в буфер возобновляется;

SR[7] – флаг «Буфер пуст», устанавливается в «1», когда в буфере отсутствуют коды клавиш (SR[3:0] = 0000).

4 – регистр данных видеопамати. Этот регистр является виртуальным, фактически запись (OUT) производится в ячейку видеопамати по адресу, установленному в Addr. Чтение (IN) из этого регистра возвращает 0.

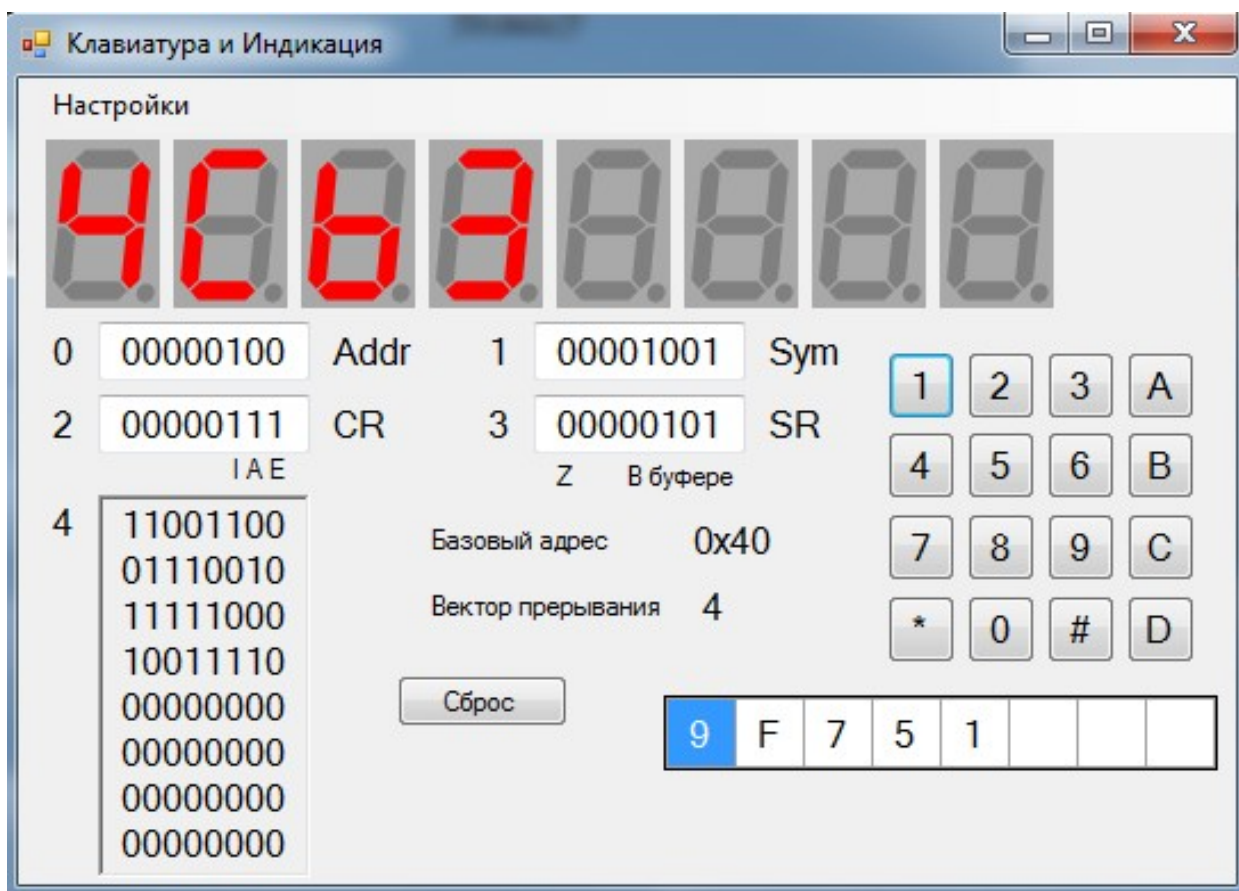


Рисунок 15. Окно обозревателя контроллера клавиатуры и индикации



Нажатие в окне обозревателя кнопки *Сброс* очищает видеопамять (и соответственно, гасит индикацию), буфер клавиатуры и все регистры. Те же действия можно реализовать программно, если записать в CR любой байт с «1» в старшем разряде или выполнить команду SBI 0x42,7.

При непустом буфере и установленном в CR флаге  $I = 1$  контроллер формирует запрос на прерывание (по умолчанию – вектор 4). Запрос снимается автоматически, когда буфер становится пуст.

### **3.6.1. Двухканальный цифровой осциллограф**

## Часть II

# Лабораторный практикум

## 4. Лабораторная работа № 1.

### Арифметические операции с многобайтовыми данными

#### 4.1. Кодирование числовых данных

Многоразрядные числа в ЭВМ могут храниться (и обрабатываться) обычно представлены в шестнадцатеричной (двоичной) или в десятичной системах счисления. При этом различают два формата представления — *упакованный* и *распакованный*.

**Упакованный формат** позволяет хранить в одном байте две шестнадцатеричные или десятичные цифры — по одной в каждой тетраде.

**Распакованный формат** при хранении как правило совпадает с ASCII-кодом соответствующего символа (цифры), причем для шестнадцатеричных цифр A..F следует выбрать заглавные или строчные коды соответствующих букв.

На рис. 16 показано, как может храниться десятичное число 382 706 в разных форматах: по адресам 0x210 .. 0x215 — распакованный формат, по адресам 0x220 .. 0x222 — упакованный формат того же числа.

**По меньшему адресу принято хранить младший фрагмент данных.**

Пример форматов шестнадцатеричных данных показан на рис. 17. Восьмиразрядное шестнадцатеричное число 4E 2F CB 05 представлено в упакованном формате по адресам 0x210 .. 0x213 (его можно рассматривать как 32-разрядное двоичное число), а по адресам 0x220 .. 0x227 — распакованный формат — ASCII-коды десятичных цифр и строчных латинских букв *a..h*. Если требуется в качестве старших шестнадцатеричных цифр использовать заглавные буквы, то в старшей тетраде кода буквы следует вместо «b» поставить «4».

В арифметических операциях распакованный формат можно преобразовать в упакованный или выполнять операции непосредственно над распакованными числами с соответствующей коррекцией.

	0	1	2	3	4	5	6	7	8
200	00	00	00	00	00	00	00	00	00
210	36	30	37	32	38	33	00	00	00
220	06	27	38	00	00	00	00	00	00

Рисунок 16. Десятичные данные в ОЗУ

	0	1	2	3	4	5	6	7	8
200	00	00	00	00	00	00	00	00	00
210	05	CB	2F	4E	00	00	00	00	00
220	35	30	62	63	66	32	65	34	00

Рисунок 17. HEX-данные в ОЗУ

## 4.2. Выполнение арифметических операций сложения и вычитания с «длинными» данными

Очевидно, если длина данных превышает длину машинного слова (в нашем случае – байта), следует последовательно выполнять заданную операцию начиная с младших байтов, обеспечивая передачу переноса (заёма) от младших байтов к старшим.

### 4.2.1. Упакованный формат

Наиболее просто выполняются операции с **упакованными шестнадцатеричными числами**. В большинстве процессоров, наряду с командами *ADD* и *SUB*, предусмотрены команды *ADDC* и *SUBC*<sup>9</sup>, которые складывают (вычитают) операнды с учётом значения флага *C*.

При отсутствии команд, автоматически учитывающих значение переноса (заёма) из предыдущего слова, необходимо обеспечить сохранность флага *C*, сформированного в текущем цикле операции, программно анализировать значение *C* в следующем цикле и при необходимости корректировать результат.

При работе с **упакованными десятичными числами** (код 8421) дополнительно требуется после операции над очередной парой байт выполнять *десятичную коррекцию* предварительного результата. Такая коррекция может выполняться специальными командами, описанными в разделе 2.3.1 настоящего пособия. При отсутствии таких команд в системе команд процессора эти же действия следует реализовать программно.

### 4.2.2. Распакованный формат

При наличии готовых процедур выполнения сложения/вычитания упакованных чисел можно упаковать распакованные данные, выполнить требуемую арифметическую операцию и, при необходимости, представить результат опять в распакованном формате. Однако, можно выполнять сложение/вычитание и непосредственно в распакованном формате.

При **сложении распакованных десятичных чисел** младшая тетрада байта предварительного результата нуждается в десятичной коррекции согласно выражению (1), стр. 16, а старшая тетрада результата всегда получается равной 'b0110' или 'b0111', учитывая, что старшие тетрады всех распакованных десятичных цифр равны 'b0011' и возможен перенос из младшей тетрады. Этот перенос должен быть передан в следующий разряд, однако флаг переноса *C* не будет сформирован. Фактически значение межразрядного переноса принимает младший бит старшей тетрады, поэтому результат после коррекции необходимо сохранить для анализа этого бита в следующем цикле сложения. После сохранения следует установить в старшей тетраде результата код 'b0011', приведя десятичную цифру результата к стандартному распакованному формату.

**Вычитание распакованных десятичных чисел** выполняется аналогично. Отличие – автоматическое формирование заёма в следующий разряд во флаге *C*.

---

<sup>9</sup>В fN8 мнемоники этих команд – *ADC* и *SUBB*.

**Арифметические операции над распакованными шестнадцатеричными числами** (ASCII-кодами цифр 0..9 и *a..f*) связаны с несколькими дополнительными проблемами.

Во-первых, младшая тетрада ASCII-кодов *a..f* принимает значения 'b0001' .. 'b0110' соответственно, поэтому перед началом арифметической операции необходимо добавить к ней 'b1001', причём такой коррекции подлежат только цифры *a..f*.

Во-вторых, старшие тетрады ASCII-кодов цифр 0..9 и *a..f* отличаются друг от друга, причём одна из них нечётная ('b0011' для 0..9), а другая – чётная ('b0110' для *a..f*<sup>10</sup>). Поэтому не удаётся определить значение межразрядного переноса по одному из разрядов старшей тетрады результата и требуется сохранить значение *AC* для добавления к следующему разряду. Кстати, значение заёма из младшей тетрады правильно переходит во флаг *C*.

В-третьих, результат арифметической операции в старшей тетраде может принимать много различных значений, например, при сложении возможны значения старшей тетрады: 0x6, 0x7, 0x9, 0xA, 0xD. Однако, в результат надо поместить одно из двух значений старшей тетрады – в зависимости от получившегося значения младшей тетрады: 'b0011' для 0..9 или 'b0110' для *a..f*<sup>10</sup>.

### 4.3. Задания

В ходе выполнения лабораторной работы необходимо разработать и отладить на ассемблере fN8 программу сложения (вычитания) многоразрядных десятичных (шестнадцатеричных) чисел произвольной разрядности, представленных как *целое без знака*. Варианты заданий (табл. 2) могут включать числа в упакованном формате в кодировке 8421 или в распакованном формате в кодировке ASCII. При этом возможен ввод операндов в заданном формате непосредственно в сегмент данных или с использованием контроллера клавиатуры. На рис. 18 – пример размещения упакованных десятичных чисел 9645012357 (по адресу 210) и 732581174 (по адресу 220) и результат сложения этих чисел по адресу 230.

	0	1	2	3	4	5	6	7	8	9
200	07	00	00	00	00	00	00	00	00	00
210	57	23	01	45	96	00	00	00	00	00
220	74	11	58	32	07	00	00	00	00	00
230	31	35	59	77	03	01	00	00	00	00
240	00	00	00	00	00	00	00	00	00	00

Рисунок 18. Десятичные упакованные операнды и результат их сложения

<sup>10</sup>Или 'b0100' для *A..F*

Если в задании предусмотрен вывод на дисплей, то как исходные операнды, так и результат должны размещаться в памяти в заданном формате и выводиться на дисплей. Рис. 19 иллюстрирует результаты операции вычитания десятичных чисел в распакованном формате, операнды вводятся с клавиатуры. И операнды и результат выводятся на дисплей.

	0	1	2	3	4	5	6	7	8	9	A
200	91	39	30	00	00	00	00	00	00	00	00
210	35	32	34	33	36	38	30	30	30	30	30
220	33	31	35	30	37	33	32	30	30	30	30
230	38	38	30	37	30	35	31	2D	30	30	30
240	00	00	00	00	00	00	00	00	00	00	00
250	00	00	00	00	00	00	00	00	00	00	00
260	00	00	00	00	00	00	00	00	00	00	00
270	32	33	37	30	35	31	33	00	00	00	00
280	00	00	00	00	00	00	00	00	00	00	00

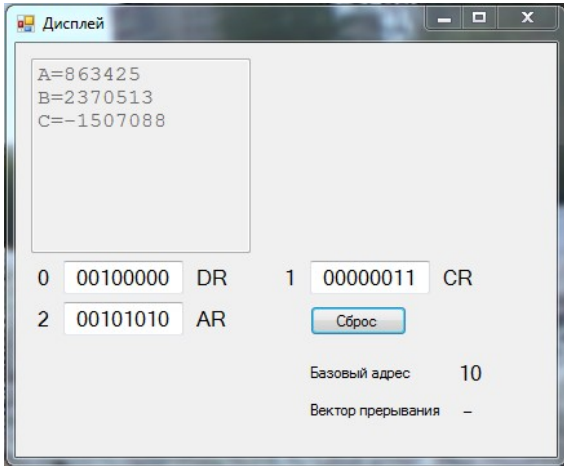


Рисунок 19. Десятичные распакованные операнды и результат вычитания

Варианты заданий приведены в табл. 2. Студент выбирает номер своего варианта в соответствии со своим порядковым номером **в списке группы**.

#### 4.4. Содержание отчёта

Отчёт выполняется в электронном виде (формат .pdf или .doc/.docx) и должен содержать:

- 1) Формулировку задания.
- 2) Граф-схемы алгоритмов основной программы и вызываемых подпрограмм.
- 3) Текст программы (и подпрограмм) на языке Ассемблера fN8 (с комментариями!).
- 4) Распределение регистров и памяти сегмента данных.
- 5) Скриншот сегмента данных с исходными операндами и результатом.
- 6) Скриншот окна обозревателя дисплея с исходными операндами и результатом, если вывод на дисплей предусмотрен заданием.

Примечание. Программа и подпрограммы (пункт 3) должны быть представлены **не рисунком, а в текстовой форме**, позволяющем копировать его в окно компилятора fN8.

Таблица 2. Варианты заданий

№ вар.	Операция	Формат	Сист. числ.	Ввод	Вывод
1	Сложение	Упаков. (8421)	DEC	Из памяти	На дисплей
2	Вычитание	Упаков. (8421)	DEC	С клавиатуры	В память
3	Вычитание	Упаков. (8421)	HEX	С клавиатуры	На дисплей
4	Сложение	Упаков. (8421)	DEC	С клавиатуры	В память
5	Вычитание	Упаков. (8421)	HEX	Из памяти	На дисплей
6	Сравнение	Упаков. (8421)	DEC	С клавиатуры	На дисплей
7	Сложение	Упаков. (8421)	HEX	С клавиатуры	В память
8	Вычитание	Распак. (ASCII)	DEC	С клавиатуры	В память
9	Сравнение	Упаков. (8421)	DEC	Из памяти	На дисплей
10	Сложение	Упаков. (8421)	DEC	С клавиатуры	На дисплей
11	Вычитание	Упаков. (8421)	HEX	С клавиатуры	В память
12	Сложение	Распак. (ASCII)	DEC	Из памяти	В память
13	Сложение	Распак. (ASCII)	DEC	Из памяти	На дисплей
14	Вычитание	Упаков. (8421)	DEC	С клавиатуры	На дисплей
15	Вычитание	Упаков. (8421)	HEX	Из памяти	В память
16	Вычитание	Распак. (ASCII)	DEC	С клавиатуры	На дисплей
17	Сравнение	Распак. (ASCII)	DEC	Из памяти	На дисплей
18	Сравнение	Распак. (ASCII)	HEX	С клавиатуры	На дисплей
19	Вычитание	Упаков. (8421)	DEC	Из памяти	На дисплей
20	Сравнение	Упаков. (8421)	HEX	Из памяти	На дисплей
21	Сравнение	Распак. (ASCII)	DEC	С клавиатуры	На дисплей
22	Сложение	Упаков. (8421)	HEX	Из памяти	На дисплей
23	Сложение	Распак. (ASCII)	DEC	С клавиатуры	В память
24	Сравнение	Распак. (ASCII)	HEX	Из памяти	На дисплей
25	Вычитание	Распак. (ASCII)	DEC	Из памяти	В память
26	Вычитание	Распак. (ASCII)	DEC	Из памяти	На дисплей
27	Сложение	Упаков. (8421)	HEX	С клавиатуры	На дисплей
28	Сложение	Распак. (ASCII)	DEC	С клавиатуры	На дисплей
29	Сравнение	Упаков. (8421)	HEX	С клавиатуры	На дисплей
30	Сложение	Упаков. (8421)	DEC	Из памяти	В память
31	Сложение	Упаков. (8421)	HEX	Из памяти	В память
32	Вычитание	Упаков. (8421)	DEC	Из памяти	В память
33	Сложение	Распак. (ASCII)	HEX	Из памяти	На дисплей
34	Сложение	Распак. (ASCII)	HEX	С клавиатуры	В память
35	Сложение	Распак. (ASCII)	HEX	С клавиатуры	На дисплей
36	Сложение	Распак. (ASCII)	HEX	Из памяти	В память
37	Вычитание	Распак. (ASCII)	HEX	С клавиатуры	В память
38	Вычитание	Распак. (ASCII)	HEX	Из памяти	В память
39	Вычитание	Распак. (ASCII)	HEX	С клавиатуры	На дисплей
40	Вычитание	Распак. (ASCII)	HEX	Из памяти	На дисплей

## 5. Лабораторная работа № 2.

### Программирование систем контроля времени

Цель лабораторной работы – разработка на fN8 группы таймеров и/или секундомеров.

Требования к секундомеру:

- Точность измерения времени – 0,1 сек.
- Диапазон отсчёта времени – 0 .. 59 мин.: 59,9 сек.
- Формат отображения состояния секундомера: мм:сс,с
- Кнопки управления:

*Пуск* – запускает отсчёт времени с текущего состояния секундомера;

*Stop* – останавливается счёт, состояние секундомера не меняется;

*Пауза* – останавливается индикация, на экран выводится состояние секундомера на момент нажатия, счёт не останавливается. Повторное нажатие восстанавливает индикацию с текущего состояния секундомера;

*Сброс* – останавливается счёт и сбрасываются все разряды секундомера.

Требования к таймеру:

- Точность измерения времени – 0,1 сек.
- Диапазон задания интервала – 00мин:01сек .. 59мин:59сек
- Формат отображения состояния таймера: мм:сс[,с]
- Кнопки управления:

*Пуск* – запускает обратный отсчёт времени от текущего состояния таймера;

*Сброс* – останавливается счёт и сбрасываются все разряды таймера;

*Цифровые клавиши* – для ввода начального значения интервала в таймер.

#### 5.1. Возможности таймерных систем микроконтроллеров и их программных моделей в fN8

В основе любой таймерной системы 8-разрядного микроконтроллера лежит двоичный 8- или 16-разрядный суммирующий (иногда – реверсивный) счётчик. На вход счётчика могут поступать по выбору счётные импульсы от одного из двух источников:

- 1) от внешнего вывода микроконтроллера; тогда таймерная система выступает в роли счётчика внешних событий. Счётные импульсы могут быть как периодическими, так и непериодическими и ограничены только верхним уровнем значения частоты;
- 2) от внутреннего источника системной тактовой частоты, обычно сниженной с помощью фиксированного или программируемого делителя. В этом случае таймерная система выполняет функции формирования или измерения временных интервалов.

В набор моделей внешних устройств fN8 входят две таймерные системы – Таймер-2 (раздел 3.4) и Таймер-5 (раздел 3.5), которые позволяют обеспечить отсчёт требуемого интервала времени.

Учитывая, что счётчики в таймерных системах микроконтроллеров работают обычно на сложение, очевидно, существует *два способа отсчёта заданного интервала времени*:

1. После очередного переполнения счётчика в него загружается константа, от которой ведётся отсчёт времени до верхнего предела счёта (переполнения). Загрузка константы может осуществляться программно или (если предусмотрен режим работы *Автозагрузка*) аппаратно, путём копирования константы из специального регистра.
2. *Сравнение (Сброс по совпадению)* – текущее значение счётчика сравнивается с константой, загруженной в специальный регистр; при совпадении значений счётчик аппаратно сбрасывается и отсчёт начинается с начала (от 0). Можно считать, что в специальном регистре задаётся произвольный верхний предел счёта.

Обнаружение факта завершения заданного интервала времени так же возможно одним из двух способов:

1. Программный анализ состояния флага, отмечающего событие *Переполнение* или *Совпадение*.
2. Обработка прерывания по соответствующему событию.

Рассмотрим возможности программных моделей таймеров в составе fN8 для формирования заданных временных интервалов и обнаружения факта их завершения.

- Таймер-2:

- В режиме 0 (*Простой счёт*) возможна программная загрузка произвольной константы, от которой будет вестись отсчёт, в счётчик TCNT. Верхний предел счёта в этом режиме всегда 0xFFFF, при переходе FFFF → 0000 устанавливается флаг переполнения OVF и формируется запрос на прерывание, если OVIE = 1.
- В режиме 3 (*Автозагрузка*) нужная константа будет загружаться в счётчик TCNT автоматически при его переполнении, если её предварительно поместить в регистр TIOR.
- В режиме 2 (*Сравнение*) в регистр TIOR помещается значение верхнего предела счёта<sup>11</sup> по достижению которого устанавливается флаг OCF (сравнения) и формируется запрос на прерывание, если OCIE = 1.

- Таймер-5:

- В режиме 0 возможна, как и в Таймере-2, программная загрузка произвольной константы в TCNT, при этом верхний предел счёта задаётся выбором одной из трёх констант (0xFFFF, 0x0FFF, 0x00FF) или содержимым регистра ICR. При достижении верхнего предела счётчик TCNT сбрасывается в

---

<sup>11</sup>Для того, чтобы при TCNT = TIOR сбрасывался TCNT, необходимо установить флаг RT = 1 в регистре управления/состояния TSCR.



- 0, устанавливается флаг переполнения OVF и формируется запрос на прерывание, если OVIE = 1.
- Возможность автозагрузки в Таймере-5 не предусмотрены.
  - В режиме 1 (Сброс по совпадению) значение TCNT сбрасывается при TCNT = OCR, устанавливается флаг совпадения OCF и формируется запрос на прерывание, если ICIE = 1. При этом, значение OCR должно быть меньше выбранного верхнего предела счёта, иначе совпадение TCNT = OCR не наступит никогда.

## 5.2. Рекомендации по выполнению лабораторной работы

### Шаг 1 – Распределение памяти

Для каждого из двух устройств следует выделить пять последовательных ячеек в сегменте данных для хранения значений десятков минут, минут, десятков секунд, секунд и десятых долей секунды. Рекомендуется располагать данные в ячейках таким образом, чтобы младшим элементам структуры данных соответствовали меньшие адреса, то есть в нашем случае в блоке из пяти соседних ячеек меньшему адресу соответствуют десятые доли секунды, а самому большому – десятки минут. Кроме двух пятиразрядных счётчиков в сегменте данных следует предусмотреть ячейки для хранения текущего состояния таймеров и секундомеров, флагов режимов и др.

### Шаг 2 – Вывод на индикацию

Выбрав ячейки ОЗУ для счётчиков, попробуем вывести их содержимое на индикацию. Следует помнить, что кроме содержимого счётчика, следует выводить его имя, например C2 03.25.9 или T 00.18.0. В ячейках счётчика хранятся десятичные цифры, причём в разрядах десятков минут и десятков секунд – не более 5. Если в качестве устройства вывода задан контроллер 7-сегментной индикации и матричной клавиатуры, то с его помощью можно индицировать только одно, активное в данный момент, устройство. Под **активным** устройством здесь понимается не то устройство, которое сейчас запущено, а то из двух, которое мы выбрали активным (с помощью специальной кнопки).

Если информация выводится на символьный дисплей, то одновременно можно выводить состояния активного и неактивного устройств, но на действия органов управления должно реагировать только активное в данный момент устройство.

При выводе цифры на индикацию её следует преобразовать к формату вывода: для 7-сегментной индикации – в семисегментный код, для символьного дисплея – в ASCII-код. Можно хранить ASCII-код и непосредственно в разрядах счётчика (в старших тетрадах разрядов заменить 0000 на 0011) и выполнять с ним арифметические операции, только надо учитывать значение старшей тетрады при сравнении текущего значения разряда с его предельными значениями. А получение семисегментного кода цифры требует табличных преобразований.

Попробуйте загрузить в счётчик произвольные цифры, написать и отладить процедуру вывода его содержимого на заданное устройство вывода.

### Шаг 3 – Органы управления

Для управления секундомером достаточно четырёх кнопок: *Пуск*, *Стоп*, *Сброс* и *Пауза*. Если в системе два секундомера, можно использовать эти кнопки только для активного в данный момент секундомера, нужно только добавить кнопку выбора активного.

Таймер управляется двумя кнопками: *Пуск* и *Сброс*, кроме того, необходимо обеспечить ввод интервала времени, который будет отсчитывать таймер. Для ввода 4-разрядного числа можно использовать цифровую клавиатуру или добавить две кнопки, независимо инкрементирующие значение минут и секунд. При этом следует учесть, что в разрядах десятков минут и десятков секунд не должны появляться цифры больше 5!

Роль кнопок могут исполнять произвольные клавиши клавиатуры компьютера или кнопки матричной клавиатуры блока *Клавиатура и индикация*. Событие *Нажатие клавиши* может контролироваться программой путём анализа значения флага готовности или с использованием механизма прерываний, причём выбрать способ может автор.

Разработайте граф-схему алгоритма, отображающую реакцию программы на нажатие используемых клавиш. Далее следует написать и отладить фрагмент программы, реализующей этот алгоритм.

### Шаг 4 – Отсчёт времени

Способ формирования отрезка времени в 0,1 сек. целиком определяется пунктами задания В, С, и D. Значение констант, загружаемых в счётчик TCNT/регистр автозагрузки или в регистр сравнения, определяется расчётом или подбором. Следует помнить, что увеличение значения константы, загружаемой в счётчик TCNT или в регистр автозагрузки уменьшает отсчитываемый отрезок времени, а увеличение константы в регистре сравнения – увеличивает.

Событие *Переполнение/Сравнение* определяется в соответствии с пунктом D задания. При этом способ определения события *Нажата клавиша* не регламентирован. С учётом пункта D задания и отсутствия во всех вариантах задания необходимости программировать динамическую индикацию, основной цикл программы может быть реализован одним из следующих вариантов:

- последовательный анализ флагов *Переполнение/Сравнение* счётчика и *Нажата клавиша* клавиатуры с вызовом соответствующих подпрограмм;
- анализ флага *Переполнение/Сравнение* и переход на обработчик прерывания по событию *Нажата клавиша*;
- анализ флага *Нажата клавиша* и переход на обработчик прерывания по событию *Переполнение/Сравнение*;
- пустой цикл с переходом на обработчики прерываний по соответствующим событиям.

### Шаг 5 – Программы INC и DEC

Для реализации функции секундомера необходимо разработать подпрограмму под

условным названием INC<sup>12</sup>, которая к пятиразрядной структуре счётчика секундомера добавляет единицу с учётом того, что первый (десятые доли секунды), второй (секунды) и четвёртый (минуты) разряды принимают значения в диапазоне 0...9, а третий и пятый (десятки секунд и минут) – в диапазоне 0...5.

Для функционирования таймера требуется подпрограмма, вычитающая единицу из такой же пятиразрядной структуры таймера с теми же ограничениями на значения разрядов.

### 5.3. Содержание отчёта

Отчёт выполняется в электронном виде (формат .pdf или .doc/.docx) и должен содержать:

- 1) Формулировку задания.
- 2) Граф-схемы алгоритмов основной программы и вызываемых подпрограмм.
- 3) Текст программы (и подпрограмм) на языке Ассемблера fN8 (с комментариями!).
- 4) Распределение регистров и памяти сегмента данных.
- 5) Несколько скриншотов устройства вывода в разных режима работы секундомеров (таймеров).

Примечание. Программа и подпрограммы (пункт 3) должны быть представлены **не рисунком, а в текстовой форме**, позволяющем копировать его в окно компилятора fN8.

### 5.4. Контрольные вопросы

- 1) Какие режимы предусмотрены в работе Таймера-2 (Таймера-5)? Какие из них можно использовать для решения задач, поставленных в Вашем индивидуальном задании? Как?
- 2) Как следует записывать (считывать) информацию в (из) 16-разрядный регистр таймера с помощью команд, оперирующих байтами?
- 3) Какими способами можно задать интервал времени, в течение которого таймер достигнет своего верхнего предела?
- 4) Как обнаружить факт достижения таймером значения верхнего предела?
- 5) Как использовать подсистему прерываний для обнаружения факта нажатия клавиши? Можно ли обнаружить этот факт без использования подсистемы прерываний?
- 6) Как определить код нажатой клавиши?
- 7) Как использовать подсистему прерываний для обнаружения факта завершения заданного для системного таймера интервала времени?
- 8) Как сформировать таблицу векторов прерываний при использовании подсистемы прерываний для обнаружения фактов нажатия клавиши и завершения заданного

---

<sup>12</sup>Рекомендуется придумать подпрограммам другие имена, не совпадающие с мнемониками команд процессора.

для системного таймера интервала времени?

## 5.5. Варианты заданий

Задание определяется следующими параметрами:

А) Система:

- 1) 2 таймера
- 2) 2 секундомера
- 3) 1 секундомер и 1 таймер

В) Счётчик времени:

- 1) Таймер\_2
- 2) Таймер\_5

С) Способ формирования временного интервала:

- 1) Сброс по совпадению
- 2) Автозагрузка или программная загрузка начального значения счёта

Д) Способ обнаружения переполнения/сравнения счётчика времени TCNT:

- 1) Анализ соответствующего флага
- 2) По прерыванию

Е) Устройство ввода:

- 1) Контроллер клавиатуры
- 2) Матричная клавиатура

Ф) Устройство вывода:

- 1) Символьный дисплей
- 2) 7-сегментный индикатор

Таблица 3. Варианты заданий

№ вар.	Задание	№ вар.	Задание	№ вар.	Задание
1	A1-B2-C1-D1-E2-F2	9	A2-B2-C2-D1-E2-F2	17	A3-B1-C2-D1-E1-F2
2	A1-B1-C1-D1-E2-F2	10	A3-B1-C2-D1-E1-F1	18	A2-B2-C2-D1-E1-F1
3	A2-B2-C1-D1-E2-F2	11	A1-B2-C2-D1-E2-F2	19	A3-B1-C2-D1-E2-F2
4	A2-B2-C1-D2-E2-F2	12	A3-B1-C2-D1-E2-F2	20	A1-B2-C1-D2-E2-F2
5	A3-B2-C2-D1-E1-F2	13	A1-B2-C1-D1-E1-F1	21	A2-B2-C2-D2-E1-F2
6	A2-B1-C2-D2-E2-F2	14	A3-B2-C2-D1-E1-F1	22	A1-B2-C2-D2-E2-F2
7	A1-B1-C2-D1-E1-F2	15	A3-B1-C1-D1-E1-F1	23	A2-B1-C2-D1-E2-F2
8	A2-B2-C2-D1-E1-F1	16	A3-B1-C2-D2-E1-F2	24	A1-B2-C1-D1-E2-F2

## **6. Лабораторная работа № 3.**

### **Широтно-импульсная модуляция**

Задача – организовать ШИМ (заданы свойства), увеличивать/уменьшать длительность [и период]

## **7. Лабораторная работа № 4.**

### **Работа с графическим дисплеем**

## Список литературы

- [1] Жмакин А. П. Архитектура ЭВМ : 2-е изд., - СПб.: БХВ-Петербург, 2010.
- [2] Основы функционирования ЭВМ: методические указания к выполнению цикла лабораторных работ / сост. А. П. Жмакин; Курск. гос. ун-т. – Курск, 2017.