Санкт-Петербургский Поли­технический Университет Петра Великого

Инс­титут компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Отчет по индивидуальному заданию

по дисциплине «Микропроцессорные системы»

«Преобразователь кодов»

|  |
| --- |
| \_\_\_\_\_\_\_\_\_\_\_  *подпись* |
| \_\_\_\_\_\_\_\_\_\_\_  *подпись* |
| \_\_\_\_\_\_\_\_\_\_\_  *подпись* |

Работу выполнил студент группы № 43501/3 Бояркин Н.С.

Работу приняли преподаватели Кузьмин А.А.

Павловский Е.Г.

Санкт-Петербург

2016

1. **Техническое задание**

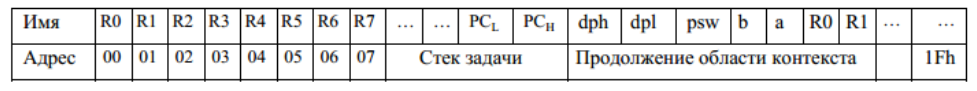
Разработать многозадачную программу, которая преобразует 8-битные числа из 2-ой, 8-ой, 10-ой, 16-ой систем счисления в 2-ю, 8-ю, 10-ю, 16-ю систему счисления по выбору пользователя. Ввод должен осуществляться с клавиатуры. Ввод неправильных значений должен обрабатываться и не вычисляться. Параллельно с вводом и преобразованием должен работать секундомер.

1. **Программа работы**
2. Преобразовать программу многозадачной операционной системы для реализации многозадачности в преобразователе кодов. Настроить секундомер и индикацию для последующего преобразования.
3. Разработать конечный автомат, который включает в себя переходы между состояниями, обработку неверных решений, побитовый ввод символов.
4. Реализовать конечный автомат в ассемблерном коде, добавить защиту от дребезга для правильной работы побитового ввода.
5. Реализовать алгоритм преобразования кода в двоичный и из двоичного в любой другой.
6. **Разработка программы**

**Преобразование программы многозадачной операционной системы**

Для реализации параллельной работы преобразователя кодов, индикации и таймера был использован код программы операционной системы из лабораторной работы «Изучение таймеров и системы прерываний».

В программе реализована многозадачность, в которой был использован принцип разделения времени: каждой задаче по очереди предоставляется свой временной промежуток, называемый квантом. Квант задается таймером, длина кванта – константой перезагрузки. Обработчику прерываний необходимо кроме перезагрузки константы таймера осуществить операцию переключения контекста, то есть сохранение набора всех регистров, используемых программой, запись их в память, вычисление адреса следующей задачи, восстановление ее контекста из памяти.

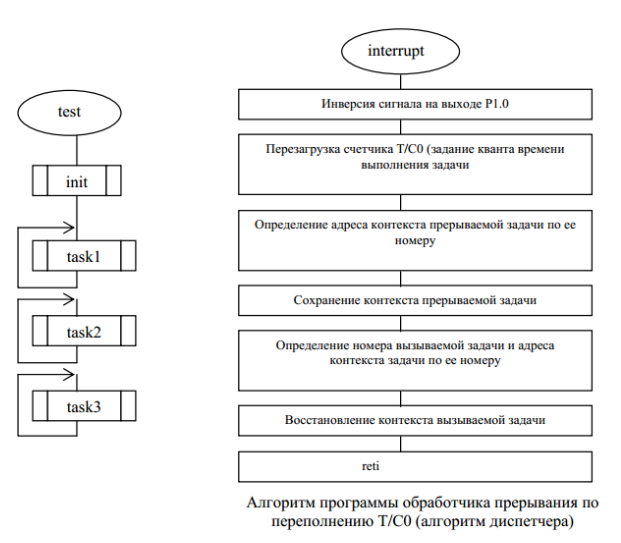


*Рис. 24. Контекст задачи*

Адрес дескриптора задачи вычисляется при помощи номера прерываемой задачи. Дескрипторы хранятся во внешней памяти, для них в программе зарезервированы области данных. Номер задачи вычисляется по следующему алгоритму. Из номера текущей задачи вычитается 2. Если при этом результат отрицательный, то значение переменной, отвечающей за номер задачи, увеличивается на единицу. Иначе ей присваивается ноль. Адреса дескрипторов отличаются только младшим байтом, чтобы проще было вычислять номер задачи.

Перед началом работы программы вызывается подпрограмма, инициализирующая таймер, устанавливающая номер вызываемой задачи в 0, инициализируются дескрипторы задач.

Каждая задача зациклена на выполнение только самой себя. При смене контекста происходит так же смена регистра PC, то есть счетчика команд. Все программы размещаются в разных областях внешней памяти, поэтому никак между собой не взаимодействуют, за исключением чтения и записи в область внутренней памяти.

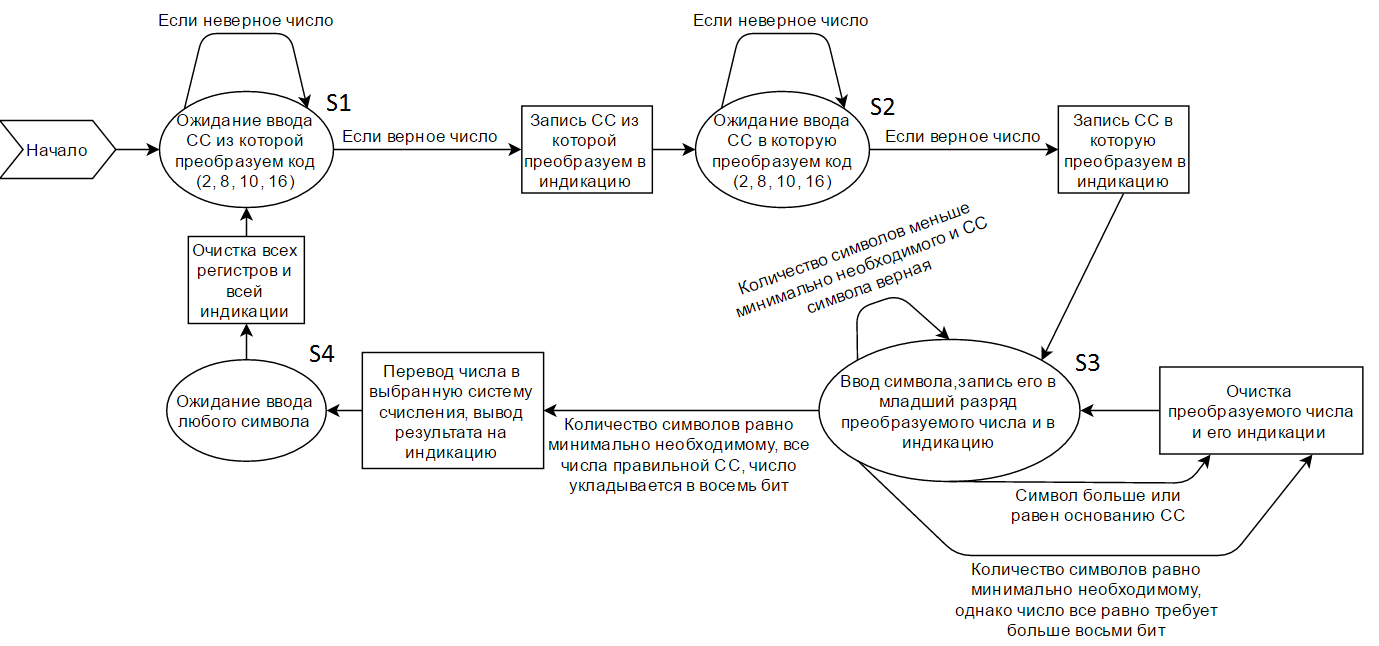


*Рис. 1. Схема алгоритма обработчика прерывания*

Первая задача – это обработчик клавиатуры и преобразователь кодов, вторая задача – индикация, третья задача – секундомер. Фактически, по сравнению с программой операционной системы, был добавлен только обработчик (конечный автомат) значения клавиатурного ввода.

**Разработка конечного автомата**

Для того, чтобы учесть все возможные комбинации пользовательского ввода с клавиатуры, при разработке программы была использована модель конечного автомата (Рис. 2). Это существенно упростило разработку программы.



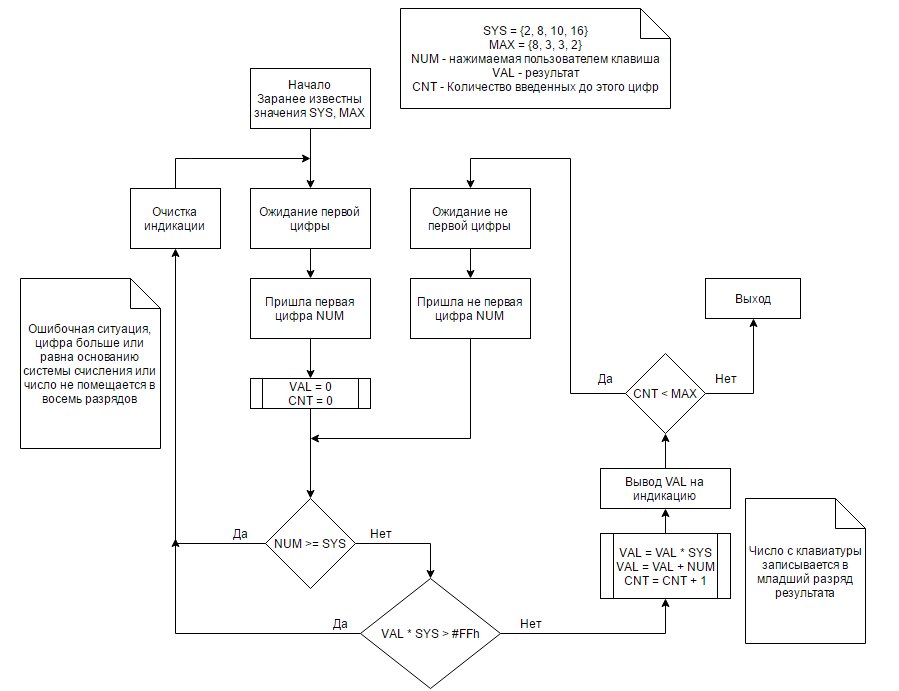
*Рис. 2. Конечный автомат обработки нажатия клавиатуры*

Конечный автомат имеет 4 состояния: S1 – ожидает ввода с клавиатуры кода системы счисления ИЗ которой мы переводим (2, 8, 10 или 16); S2 – ожидает ввода с клавиатуры кода системы счисления В которую мы переводим (2, 8, 10 или 16); S3 – осуществляет ввод числа по одной цифре в его системе счисления; S4 – вырожденное состояние, необходимое для задержки значения на экране, которое ничего не делает.

Таким образом осуществляется два преобразования: из вводимого по одной цифре преобразуемого числа в двоичную форму и из двоичной формы в результирующую систему счисления.

**Разработка алгоритма перевода в двоичное представление**

Алгоритм перевода в двоичное представление – это состояние S3 в алгоритме конечного автомата (Рис. 2). Как видно по структуре конечного автомата, на момент состояния S3 уже известна система счисления из которой (FRMSYS) мы будем переводить число. Также известно количество символов, которое должно быть введено для окончания алгоритма (MAX). Например, для того чтобы заполнить 8 разрядов в двоичной системе необходимо MAX = 8 разрядов, а для восьмеричной системы счисления необходимо всего MAX = 3 разряда. Переменные FRMSYS и MAX являются начальными данными для алгоритма.



*Рис. 3. Алгоритм перевода в двоичное представление*

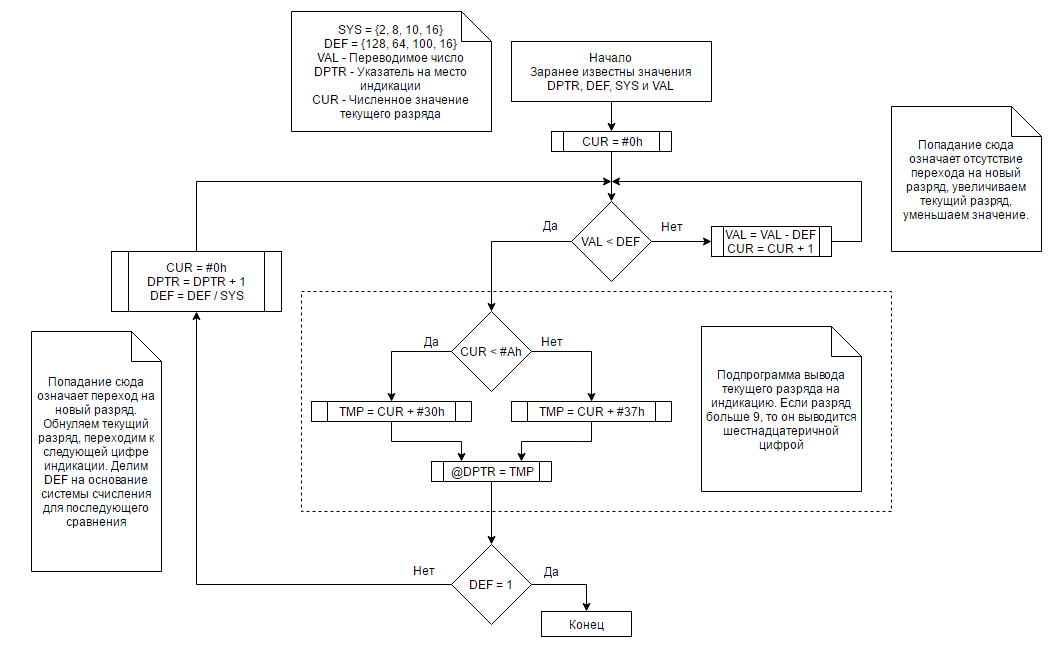
Как видно из алгоритма (Рис. 3) ошибочные ситуации возникают, когда пользователь вводит число большее основания системы счисления или, когда число не помещается в восемь разрядов. В этих случаях очищается индикация числа, и оно может быть введено заново.

Если введенное с клавиатуры число NUM верное, то результат VAL умножается на основание системы счисления FRMSYS и прибавляется NUM. Например, основание системы счисления FRMSYS = 8 и пользователь вводит 5. Так как это первая цифра . После этого пользователь вводит 2. Значение . Таким образом число восьмеричной системы счисления было записано в двоичный восьмибитный регистр.

**Разработка алгоритма перевода в произвольную СС**

Алгоритм перевода в произвольную систему счисления – это блок кода между состояниями S3 и S4 в алгоритме конечного автомата (Рис. 2). Как видно по структуре конечного автомата, на момент состояния S3 уже известна система счисления в которую (TOSYS) мы будем переводить число. Также известно максимальное значение, которое является степенью системы счисления, однако не выходит за рамки восьми разрядов (DEF). Например, значение TOSYS = 10, тогда максимальная степень числа десять, которая не выходит за рамки восьми разрядов – это степень 2 (). Системам счисления TOSYS={2, 8, 10, 16} соответствуют значения DEF={128, 64, 100, 16}. Из структуры конечного автомата также очевидно, что состояние S3 (Рис. 3) уже прошло, а это значит, что двоичное представление VAL уже сформировано. Указатель на место в памяти для индикации DPTR, также известен.

На основании этих входных данных был сформирован алгоритм перевода в произвольную систему счисления с последующим выводом на индикацию:



*Рис. 4. Алгоритм перевода в произвольную СС*

Этот алгоритм не предусматривает обработку ошибочных ситуаций, так как все входные данные для алгоритма не могут быть неверными. Вывод на индикацию поддерживает также шестнадцатеричный формат.

Приведем пример работы подпрограммы при VAL = 10, TOSYS = 8, DEF = 64:

|  |  |  |
| --- | --- | --- |
| ИТЕРАЦИЯ 1  CUR = 0  VAL = 10  DEF = 64  VAL < DEF  **ВЫВОД CUR = 0**  CUR = 0  DPTR++  DEF = 64 / 8 = 8  ИТЕРАЦИЯ 4  CUR = 0  VAL = 2  DEF = 1  VAL > DEF  VAL = 2 – 1 = 1  CUR = 0 + 1 = 1 | ИТЕРАЦИЯ 2  CUR = 0  VAL = 10  DEF = 8  VAL > DEF  VAL = 10 – 8 = 2  CUR = 0 + 1 = 1  ИТЕРАЦИЯ 5  CUR = 1  VAL = 1  DEF = 1  VAL > DEF  VAL = 1 – 1 = 0  CUR = 1 + 1 = 2 | ИТЕРАЦИЯ 3  CUR = 1  VAL = 2  DEF = 8  VAL < DEF  **ВЫВОД CUR = 1**  CUR = 0  DPTR++  DEF = 8 / 8 = 1  ИТЕРАЦИЯ 6  CUR = 2  VAL = 0  DEF = 1  VAL < DEF  **ВЫВОД CUR = 2**  ВЫХОД (DEF = 1) |

Таким образом число из восьмибитного регистра было выведено на индикацию (1010 = 0128).

**Защита от дребезга**

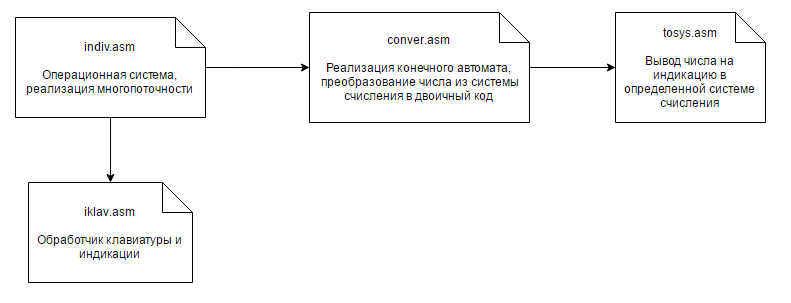
Защита от дребезга была реализована с помощью программной задержки, которая вызывается после нажатия на кнопку:

|  |
| --- |
| 1. kdelay: 2. **mov** r0, #0h *; Младший байт* 3. **mov** r1, #70h *; Старший байт* 4. dlyloop: 5. **inc** r0 6. **mov** a, r0 7. cjne a, #0h, dlyloop 8. **inc** r1 9. **mov** a, r1 10. cjne a, #0h, dlyloop 11. **ret** |

Шестнадцатеричное число инкрементируется в цикле до переполнения. Длительность задержки можно регулировать, задавая начальные значения этого числа.

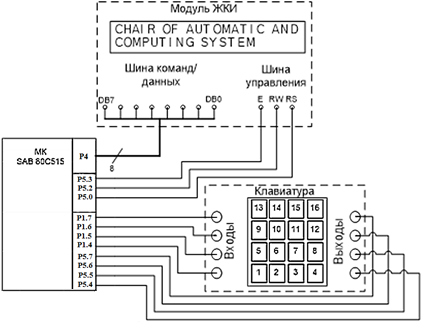
**Структура проекта и код программы**

Проект содержит четыре файла, которые организованы следующим образом:



*Рис. 5. Структура проекта*

Перед запуском программы были подключены ЖКИ и клавиатура, согласно схеме соединений (Рис. 6).



*Рис. 6. Схема соединений*

Код основного модуля программы (indiv.asm):

|  |
| --- |
| *; indiv.asm*  org 8100h    num\_task: **equ** 50h    **mov** **SP**, #7h      lcall init    *; Задача1 - определение номера нажатой клавиши*  *; и преобразование в ASCII код*  prog1:      cpl P1.1      lcall memklav      lcall gloop    **mov** a, 34h  **add** a, #0h  **jz** prog1    **mov** a, 34h      clr c      subb a, #FFh  **jz** prog1        lcall kdelay      sjmp prog1    *; Задача 2 - индикация номера нажатой клавиши*  prog2:      cpl P1.2      lcall indic      sjmp prog2    *; Задача 3 - электронные часы*  prog3:      cpl P1.3      lcall clock      sjmp prog3    *; дескриптор задачи1*  prog1\_d:    **db** 11h, 1,  0, 0, 0, 0, 0, 0, 00, 00, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0    *; дескриптор задачи2*  prog2\_d:    **db** 11h, 1,  0, 0, 0, 0, 0, 0, 00, 00, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0    *; дескриптор задачи3*  prog3\_d:    **db** 11h, 1,  0, 0, 0, 0, 0, 0, 00, 00, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0    *; инициализация дескрипторов и таймера*  init:      lcall initsys  **mov** num\_task, #0  *; обнуление декод. значения клавиши*  **mov** 34h, #0    **mov** dptr, #prog2  **mov** r0, DPL  **mov** r1, DPH  **mov** dptr, #prog2\_d      lcall init\_rout    **mov** dptr, #prog3  **mov** r0, DPL  **mov** r1, DPH  **mov** dptr, #prog3\_d      lcall init\_rout    *;anl TMOD, #00000000b*  *;orl TMOD, #00000001b*  **mov** TMOD, #11110001b  **mov** TH0, #F6h *; fe*  **mov** TL0, #3Bh *; 39*    *;сброс регистров значений таймера*  **mov** ms, #0h  **mov** sd10, #0h  **mov** sec, #0h  **mov** min, #0h  **mov** hours, #0h    **setb** ea *;разрешение всех прерываний*  **setb** et0    *;разрешение прерываний от таймера 0*  **setb** tr0    *;разрешение счета таймера 0*  **ret**    init\_rout:  **mov** a, DPL  **add** a, #8  **mov** DPL, a    **mov** a, r0      movx @dptr, a  **inc** dptr  **mov** a, r1      movx @dptr, a  **ret**    tim0:  **mov** TH0, #F6h   *; 2,5 ms F6 47*  **mov** TL0, #47h        cpl p1.0    *; программа-диспетчер*  dispatcher:    *; сохранение SFR*  **push** dph  **push** dpl  **push** psw  **push** b  **push** a  **push** 0  **push** 1        lcall def\_context\_addr    *; сохранение контекста*  **mov** r0, **sp**  *;количество сохраняемых параметров*  **inc** r0  **mov** r1, #0h  prpm1:  **mov** a, @r1      movx @dptr, a  **inc** r1  **inc** dptr      djnz r0, prpm1        lcall def\_ntx\_tsk      lcall def\_context\_addr        movx a, @dptr  **mov** b, a  **mov** r1, #0h  prpm2:      movx a, @dptr  **mov** @r1, a  **inc** r1  **inc** dptr      djnz r0, prpm2    *; восстановление SFR*  **dec** b  **mov** **SP**, b  **pop** 1  **pop** 0  **pop** a  **pop** b  **pop** PSW  **pop** DPL  **pop** DPH        reti    *; определение DPTR*  def\_ntx\_tsk:  **inc** num\_task  **mov** a, num\_task      cjne a, #3h, exit\_def\_ntf  **mov** num\_task, #0h  exit\_def\_ntf:  **ret**    def\_context\_addr:  **mov** a, num\_task      rr a      rr a      rr a  **mov** r0, a  **mov** dptr, #prog1\_d  **mov** a, DPL  **add** a, r0  **mov** DPL, a  **jnc** exit\_def  **inc** DPH  exit\_def:  **ret**    kdelay:  **mov** r0, #0h  **mov** r1, #70h  dlyloop:  **inc** r0  **mov** a, r0      cjne a, #0h, dlyloop  **inc** r1  **mov** a, r1      cjne a, #0h, dlyloop  **ret**        include ASMS\43501\_3\bk\b5\clock.asm      include ASMS\43501\_3\bk\b5\conver.asm      include ASMS\43501\_3\bk\b5\tosys.asm      include ASMS\43501\_3\bk\b5\iklav.asm        org 800bh      ljmp tim0 |

Код подключаемого модуля конечного автомата (conver.asm):

|  |
| --- |
| *; conver.asm*    *; DEF, FRMSYS, TOSYS, VAL*  def:    **equ** 60h  frmsys: **equ** 61h  tosys:  **equ** 62h  value:  **equ** 63h    current:    **equ** 64h    *; Состояния S1 S2 S3*  sfrm:   **equ** 65h  sto:    **equ** 66h  swait:  **equ** 67h    *; CNT, MAX*  vcount: **equ** 68h  mcount: **equ** 69h      initsys:  *; Установка состояния S1*  **mov** sfrm, #1h  **mov** sto, #0h  **mov** swait, #0h    **mov** vcount, #0h  **mov** mcount, #0h  **ret**    gloop:  **mov** a, 34h  **add** a, #0h  **jz** exgloop      clr c      subb a, #FFh  **jz** exgloop    *; Если состояние S1*  **mov** a, sfrm      clr c      subb a, #1h  **jnz** nextto      lcall hfrm  **jmp** exgloop    nextto:  *; Если состояние S2*  **mov** a, sto      clr c      subb a, #1h  **jnz** nextwait      lcall hto  **jmp** exgloop    nextwait:  *; Если состояние S3*  **mov** a, swait      clr c      subb a, #1h  **jnz** lastsys      lcall hwait  **jmp** exgloop      lastsys:  *; Перевод в систему счисления и вывод на индикацию*      lcall tosyshdr      lcall initsys    exgloop:  **ret**    *;;;;;;;;;;;;;;;;;;;;;;;;;;*    *; Обработчик состояния S1*    hfrm:      lcall clrall    **mov** a, 34h      clr c      subb a, #2  **jnz** frmnext8  **mov** frmsys, #2  **mov** mcount, #8  **jmp** chfrm    frmnext8:  **mov** a, 34h      clr c      subb a, #8  **jnz** frmnext10  **mov** frmsys, #8  **mov** mcount, #3  **jmp** chfrm    frmnext10:  **mov** a, 34h      clr c      subb a, #10  **jnz** frmnext16  **mov** frmsys, #10  **mov** mcount, #3  **jmp** chfrm    frmnext16:  **mov** a, 34h      clr c      subb a, #16  **jnz** exfrm  **mov** frmsys, #16  **mov** mcount, #2  **jmp** chfrm    chfrm:  **mov** sfrm, #0h  **mov** sto, #1h  **mov** swait, #0h      lcall drwfrm*;*  exfrm:  **ret**    *;;;;;;;;;;;;;;;;;;;;;;;;;;*    *; Обработчик состояния S2*    hto:  **mov** a, 34h      clr c      subb a, #2  **jnz** tonext8  **mov** tosys, #2  **mov** def, #128  **jmp** chto    tonext8:  **mov** a, 34h      clr c      subb a, #8  **jnz** tonext10  **mov** tosys, #8  **mov** def, #64  **jmp** chto    tonext10:  **mov** a, 34h      clr c      subb a, #10  **jnz** tonext16  **mov** tosys, #10  **mov** def, #100  **jmp** chto    tonext16:  **mov** a, 34h      clr c      subb a, #16  **jnz** exto  **mov** tosys, #16  **mov** def, #16  **jmp** chto    chto:  **mov** sfrm, #0h  **mov** sto, #0h  **mov** swait, #1h      lcall drwto*;*  exto:  **ret**    *;;;;;;;;;;;;;;;;;;;;;;;;;;*    *; Обработчик состояния S3*  *; Алгоритм перевода в двоичное представление*    hwait:  **mov** a, 34h  **dec** a      clr c      subb a, frmsys  **jc** suwait  **jmp** badres    suwait:  **mov** a, value  **mov** b, frmsys  **mul** ab  **mov** value, a  **mov** a, b  **add** a, #0h  **jnz** badres  **mov** a, value  **dec** a  **add** a, 34h  **mov** value, a  **inc** vcount      lcall drwwait      clr c  **mov** a, vcount      subb a, mcount  **jnz** exwait  **mov** sfrm, #0h  **mov** sto, #0h  **mov** swait, #0h  **jmp** exwait    badres:  **mov** value, #0h  **mov** vcount, #0h      lcall clrval    exwait:  **ret**    *;;;;;;;;;;;;;;;;;;;;;;;;;;*    *; Индикация состояния S1*    drwfrm:  **mov** dptr, #str1 + 14    **mov** a, frmsys      clr c      subb a, #10  **jc** seccrf  **mov** a, #31h      movx @dptr, a  **mov** a, frmsys      clr c  **add** a, #26h  **jmp** secsmf    seccrf:  **mov** a, frmsys  **add** a, #30h    secsmf:  **inc** dptr      movx @dptr, a  **ret***;*    *;;;;;;;;;;;;;;;;;;;;;;;;;;*    *; Индикация состояния S2*    drwto:  **mov** dptr, #str1 + 18    **mov** a, tosys      clr c      subb a, #10  **jc** seccrt  **mov** a, #31h      movx @dptr, a  **mov** a, tosys      clr c  **add** a, #26h  **jmp** secsmt    seccrt:  **mov** a, tosys  **add** a, #30h    secsmt:  **inc** dptr      movx @dptr, a  **ret**      *;;;;;;;;;;;;;;;;;;;;;;;;;;*    *; Индикация состояния S3*    drwwait:  **mov** dptr, #str2    **mov** r0, #0h  drwwloop:  **inc** r0  **inc** dptr  **mov** a, r0      cjne a, vcount, drwwloop    **mov** a, 34h  **dec** a    **mov** r0, a        clr c      subb a, #Ah  **mov** a, r0  **jc** drwwnum  **add** a, #7h  drwwnum:  **add** a, #30h        movx @dptr, a    **ret**    *;;;;;;;;;;;;;;;;;;;;;;;;;;*    *; Очистка индикации переводимого значения*    clrval:  **mov** dptr, #str2 + 1  **mov** r0, #8h  clrvsl:  **dec** r0  **mov** a, #20h      movx @dptr, a  **inc** dptr  **mov** a, r0      cjne a, #0h, clrvsl  **ret**      *;;;;;;;;;;;;;;;;;;;;;;;;;;*    *; Очистка всей индикации*    clrall:  **mov** dptr, #str1 + 14  **mov** a, #20h      movx @dptr, a  **inc** dptr  **mov** a, #20h      movx @dptr, a    **mov** dptr, #str1 + 18  **mov** a, #20h      movx @dptr, a  **inc** dptr  **mov** a, #20h      movx @dptr, a    **mov** dptr, #str2 + 1  **mov** r0, #8h  clrv:  **dec** r0  **mov** a, #20h      movx @dptr, a  **inc** dptr  **mov** a, r0      cjne a, #0h, clrv    **mov** dptr, #str2 + 12  **mov** r0, #8h  clrr:  **dec** r0  **mov** a, #20h      movx @dptr, a  **inc** dptr  **mov** a, r0      cjne a, #0h, clrr    **ret** |

Код подключаемого модуля алгоритма перевода в произвольную СС (tosys.asm):

|  |
| --- |
| *; tosys.asm*  *; Алгоритм перевода в произвольную СС*    tosyshdr:  *; Инициализация DPTR, CUR*  *; Остальные значения уже известны*  **mov** current, #0h  **mov** dptr, #str2 + 12      lcall clrmem  **mov** dptr, #str2 + 12    **start**:  **mov** a, value      clr c      subb a, def  **jnc** incnum      lcall toascii      movx @dptr, a  **mov** a, def      clr c      subb a, #1h  **jz** exit  **mov** current, #0h  **inc** dptr  **mov** a, def  **mov** b, tosys  **div** ab  **mov** def, a  **jmp** **start**  incnum:  **mov** value, a  **inc** current  **jmp** **start**  exit:  **ret**    *; Вывод цифры на экран*    toascii:  **mov** a, current      clr c      subb a, #Ah  **mov** a, current  **jc** decnum  **add** a, #7h  decnum:  **add** a, #30h  **ret**    *; Очищение индикации*    clrmem:  **mov** r0, #8h  clrloop:  **dec** r0  **mov** a, #20h      movx @dptr, a  **inc** dptr  **mov** a, r0      cjne a, #0h, clrloop  **ret** |

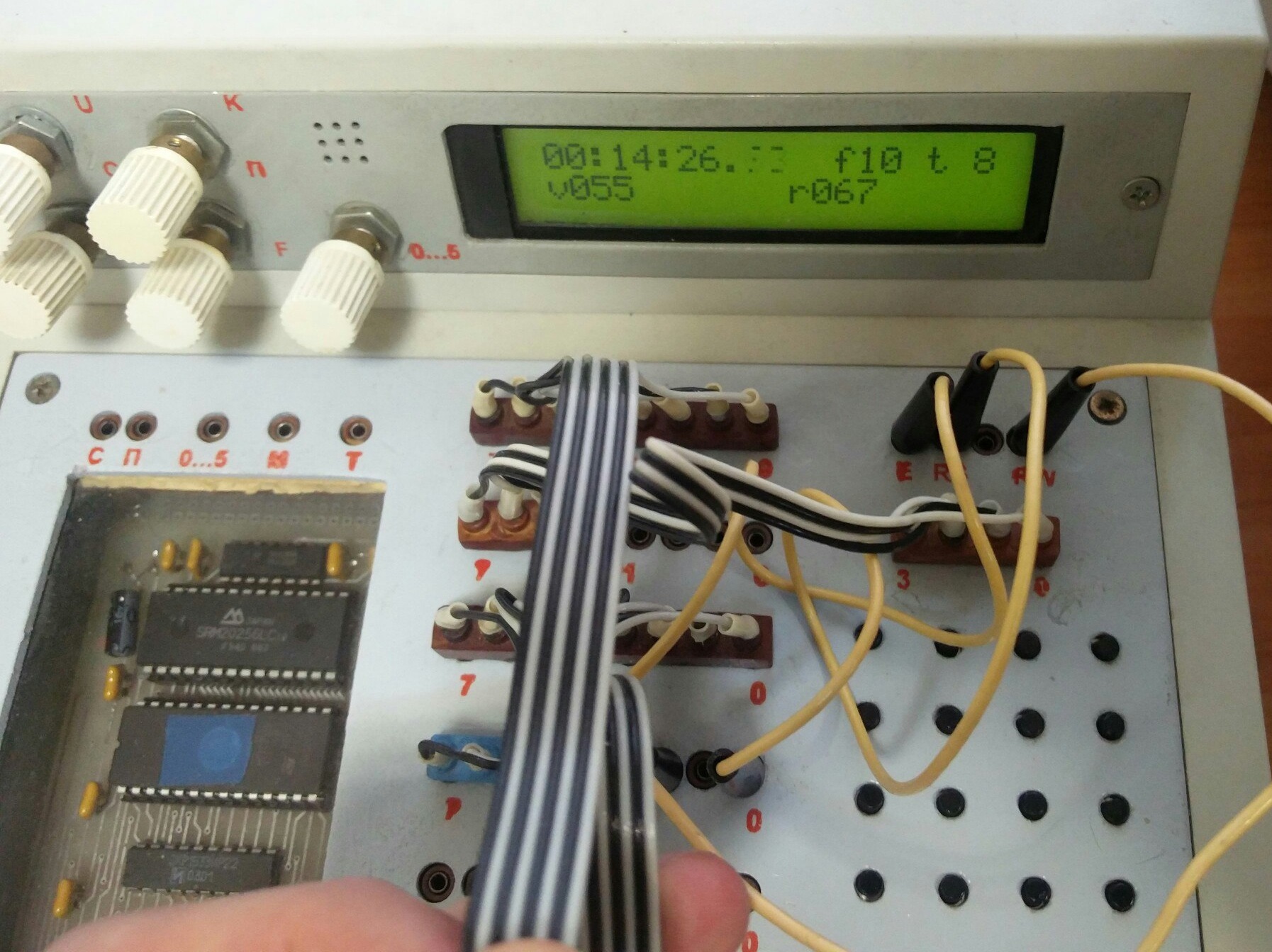
Код подключаемого модуля индикации и клавиатуры (iklav.asm):

|  |
| --- |
| *; iklav.asm*      org 8700h    P4:     **equ** E8h  P5: **equ** F8h    indic:  clr P5.0        *; Подготовка к вводу КОМАНД: RS = 0*  **mov** r4, #38h    *; 8-битовый режим обмена с выводом обеих строк*      lcall ind\_wr    *; Запись команды в ЖКИ*  **mov** r4, #0Ch    *; Активизация всех знакомест дисплея без курсора*      lcall ind\_wr  **mov** r4, #80h    *; Адрес нулевой ячейки 1-ой строки*      lcall ind\_wr    **mov** dptr, #FFD0h  **setb** P5.0   *; Подготовка к вводу ДАННЫХ: RS = 1*    *; Выводим 1-ую строку*  wr\_str1:    movx a, @dptr *; Читаем символ из внешней памяти*  **mov** r4, a      lcall ind\_wr  *; Запись данных в ЖКИ*  **inc** dptr      *; Формируем сл. адрес видеобуфера*  **mov** a, dpl  *; Мл. часть dptr*      cjne a, #E4h, wr\_str1*; Проверка окончания вывода символов 1 строки*        clr P5.0  **mov** r4, #C0h      lcall ind\_wr  **setb** P5.0    *; Выводим 2-ую строку*  wr\_str2:    movx a, @dptr *; Читаем символ из внешней памяти*  **mov** r4, a     *; Запись данных в ЖКИ*      lcall ind\_wr  *; Формируем сл. адрес видеобуфера*  **inc** dptr  **mov** a, dpl      cjne a, #0F8h, wr\_str2*; Проверка окончания вывода символов 2 строки*  **ret**    ind\_wr:     **mov** P4, r4 *; Грузим в порт Р4 передаваемую посылку*  **setb** p5.3   *; Установка сигнала Е*      clr p5.2    *; Сигнал R/W=0 (запись)*      lcall delay      clr p5.3    *; Сброс сигнала Е*      lcall delay  **setb** p5.3  **ret**    delay:  **mov** r3, #7  m2: djnz r3, m2  **ret**    memklav:  **mov** 20h, #0h *; 0 for clear C*  **mov** R1, #33h    *; Адрес первой ячейки памяти для просмотра*  **mov** R3, #3h *; счетчик(по строкам и столбцам)*  **mov** 35h, #0h    *; Счётчик нажатых клавиш*  **mov** 37h, #0h    *; Код символа*  **mov** 38h, #0h    *; номер строки*  **mov** 39h, #0h    *; номер столбца*      lcall klav    *; Сначала - проверка на ноль (ничего не нажато)*  zero\_chk:  **mov** C, 0h       *; clear C*  **mov** A, @R1  *; Читаем данные из памяти*  *;mov 56h, R1*      subb A, #f0h    *; Отнимаем 0Fh - если будет ноль, то ничего не нажато.*  *; Иначе считаем, что было какое-нибудь нажатие.*  **jz** skip\_cntr    *; A==0 - пропускаем счётчик нажатий*  **inc** 35h     *; Не ноль - инкремент счётчика нажатий*  **mov** A,@R1  **mov** 37h,A       *; Сохраняем код нажатой клавиши.*  **mov** 38h,R3  *; Сохранили номер строки нажатой клавиши*    skip\_cntr:  **dec** R1      *; Берём следующий элемент из памяти*  *; Пока не достигли конца массива для проверки -*  **dec** R3      *; увеличиваем номер строки*  **mov** C, 0h       *; clear C*      cjne R1, #2Fh, zero\_chk *; - продолжаем цикл*  *; Вышли из цикла проверки отсутствия нажатий*      **mov** A, 35h  *; Грузим в А счётчик нажатий*  **jz** wr\_0     *; 0 нажатий - пишем ноль*  **mov** C, 0h       *; clear C*      cjne A, #01h, wr\_FF     *; больше 1 нажатия - пишем FF*    **mov** dptr, #cdMask   *; начало массива кодов*  **mov** R3,#0h*; ; обнулили счетчик*    find\_column:  **inc** R3*;     ; счетчик номера столбца*  **mov** 39h,R3  *; сохраняем номер столбца*  **mov** A,R3*;*  **mov** C, 0h       *; clear C*      subb A,#5h  **jz** wr\_FF *; Т.к. клавишу точно нажали(или несколько)*  *; ее код обазятельно должен найтись в массиве*  *; иначе - было нажато несколько клавиш, и код не совпал*      movx A, @dptr   *; записали элемент*  **inc** dptr        *; сразу inc индекс в массиве*  **mov** C, 0h       *; clear C*      cjne A, 37h, find\_column *; если число не равно найденному,*  *; продолжим поиск*  get\_num:  *; номер строки\*4+номер столбца*  **mov** A, 38h  **mov** C, 0h       *; clear C*      rl A      rl A            *; два сдвига числа =\*4*  *;add A, 39h     ; получили число*  **add** A, #5h      subb A, 39h  **mov** 34h, A  *; запись числа*      sjmp ext  wr\_0:   **mov** 34h, #0h      sjmp ext  wr\_FF:  **mov** 34h, #FFh      sjmp ext    *; Существующие коды клавиш - характерны для столбца.*  cdMask: **db** E0h, D0h, B0h, 70h    ext:    **ret**    p5: **equ** f8h    klav:   **mov** r0, #30h    *; задаем адрес карты памяти*      orl p5, #f0h    *; настраиваем порт на ввод*  **mov** a, #7fh *; загружаем код бегущего нуля*    mb: **mov** r2, a        rlc a  **mov** p1.7, c      rlc a  **mov** p1.6, c      rlc a  **mov** p1.5, c      rlc a  **mov** p1.4, c    **mov** a, p5   *; считываем данные с клавиатуры*      anl a,#f0h  **mov** @r0, a  *; и запоминаем их*  **inc** r0      *; увеличиваем адрес для записи*  **mov** a, r2      rr a        *; осуществляем сдвиг*      cjne a, #f7h, mb*; выполняем цикл*  **ret**    *;Приведение полученной цифры к десятичному формату*  decim:  **mov** a,34h      cjne a,#ffh, wrff  **mov** a,#46h  **mov** dptr, #str2 + 17      movx @dptr,a  **inc** dptr      movx @dptr,a  **ret**    wrff:   **mov** dptr, #str2 + 17  **mov** a, 34h  **mov** b, #10  **div** ab  **add** a, #30h      movx @dptr, a  **inc** dptr  **mov** a, b  **add** a, #30h      movx @dptr, a  **ret**    *; видеобуффер*          org FFD0h  str1:   **db**  20h, 20h, 20h, 3Ah, 20h, 20h, 20h, 3Ah, 20h, 20h, 20h, 2eh, 20h, 20h, 30h, 30h, 20h, 20h, 20h, 20h  str2:   **db**  'BUTTON NUMBER:      ' |

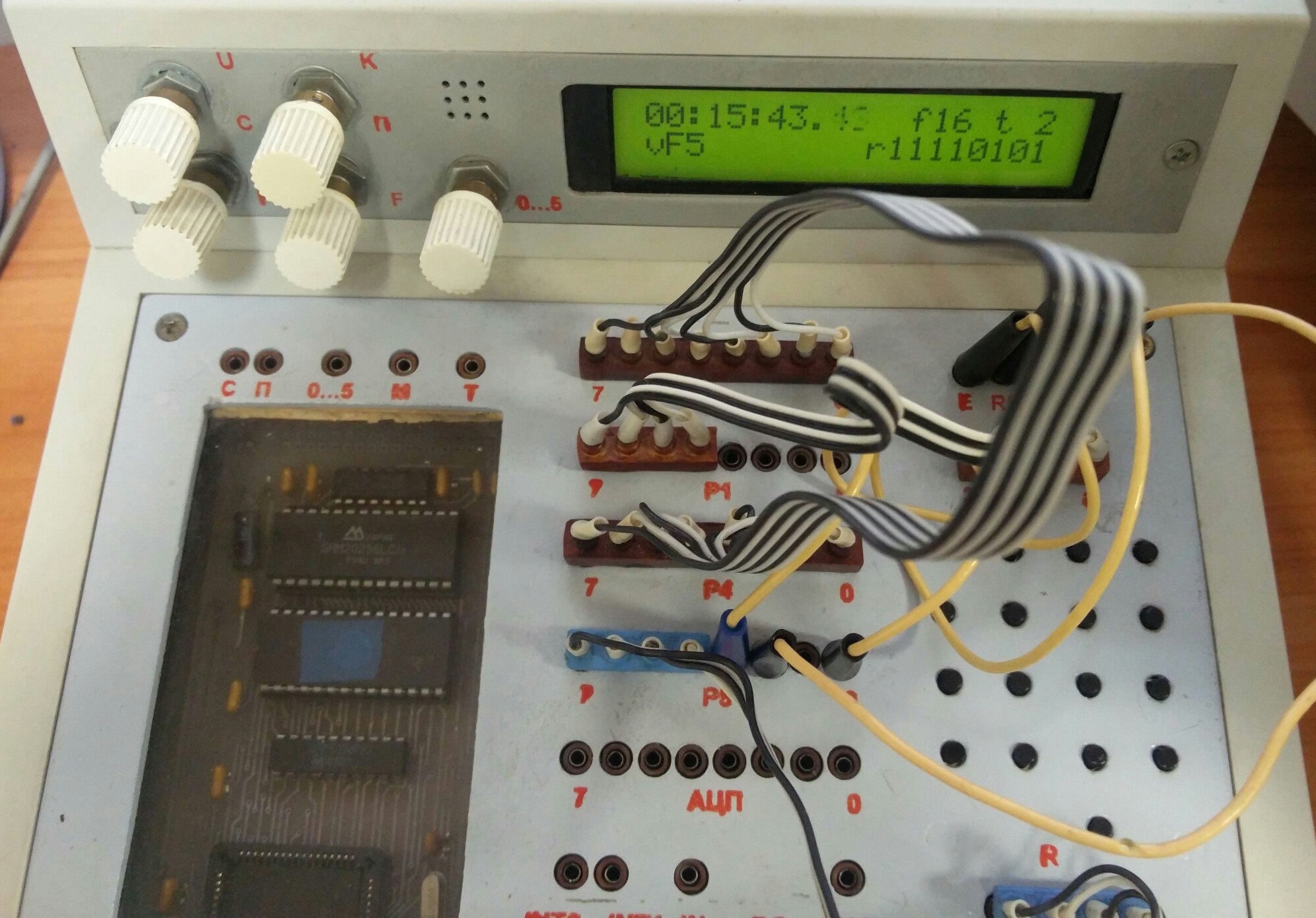
Код подключаемого модуля таймера (clock.asm):

|  |
| --- |
| *; clock.asm*      org 8300h    ms: **equ** 3Fh *;тики*  sd10:   **equ** 40h *;100 мс*  sec:    **equ** 41h *;с*  min:    **equ** 42h *;м*  hours:  **equ** 43h *;ч*    *; »нициализаци¤*  initt:  *;orl TMOD, #00010000b ;дл¤ работы в режиме 16-битного счЄтчика*        anl TMOD, #1Fh  **mov** TH1, #FEh   *;инициализаци¤ счЄтчика T/C1 дл¤*  **mov** TL1, #0Bh   *;формировани¤ "тика" 5 мс*    *; наивысший приоритет дл¤ T/C1*  **mov** A9h, #08h  **mov** B9h, #00h    *;setb ea    ;разрешение всех прерываний*  **setb** et1    *;разрешение прерывани¤*  **setb** tr1    *;разрешение счЄта*  **ret**  clock:    tim1:   **mov** TH1, #FEh    *;2KHz = 500mks*  **mov** TL1, #17h    *; 17*    **inc** ms       *;инкремент тиков*  **mov** r5, ms      clr c  **mov** a, r5      subb a, #200  **jc** end\_tim  *;если количество тиков равно 200*  **mov** ms, #0h      lcall inc\_dec\_sec    end\_tim:    reti    *; »нкрементируем мс, сек и мин*  inc\_dec\_sec:  **inc** sd10    *; инкремент 0,1 сек*  **mov** r5, sd10      cjne r5, #64h, end  *; проверка сек == 0,1*  **inc** sec     *; инкремент секунд*  **mov** r5, sec  **mov** sd10, #0h      cjne r5, #3Ch, end  *; проверка сек == 60*  **inc** min     *; инкремент минуты*  **mov** r5, min  **mov** sec, #0h      cjne r5, #3Ch, end  *; проверка мин == 60*  **inc** hours       *; инкремент часы*  **mov** min, #0h    end:    lcall to\_int  **ret**        org 801bh      ljmp tim1        org 8500h    to\_int:    *; ƒл¤ дес¤тых долей секунд*  **mov** a, 40h  **mov** dptr, #FFDAh      lcall overal    **dec** dpl      lcall overal    *; ƒл¤ секунд*  **mov** a, 41h  **mov** dptr, #FFD7h      lcall overal    **dec** dpl      lcall overal    *; ƒл¤ минут*  **mov** a, 42h  **mov** dptr, #FFD4h      lcall overal    **dec** dpl      lcall overal    *; ƒл¤ часов*  **mov** a, 43h  **mov** dptr, #FFD1h      lcall overal    **dec** dpl      lcall overal  **ret**    overal: **mov** b, #10d *;основание системы счислени¤*  **div** ab  **mov** r1, a  **mov** a, b  **add** a, #30h *;ASCII символа*      movx @dptr, a   *;символ*  **mov** a, r1  **ret** |

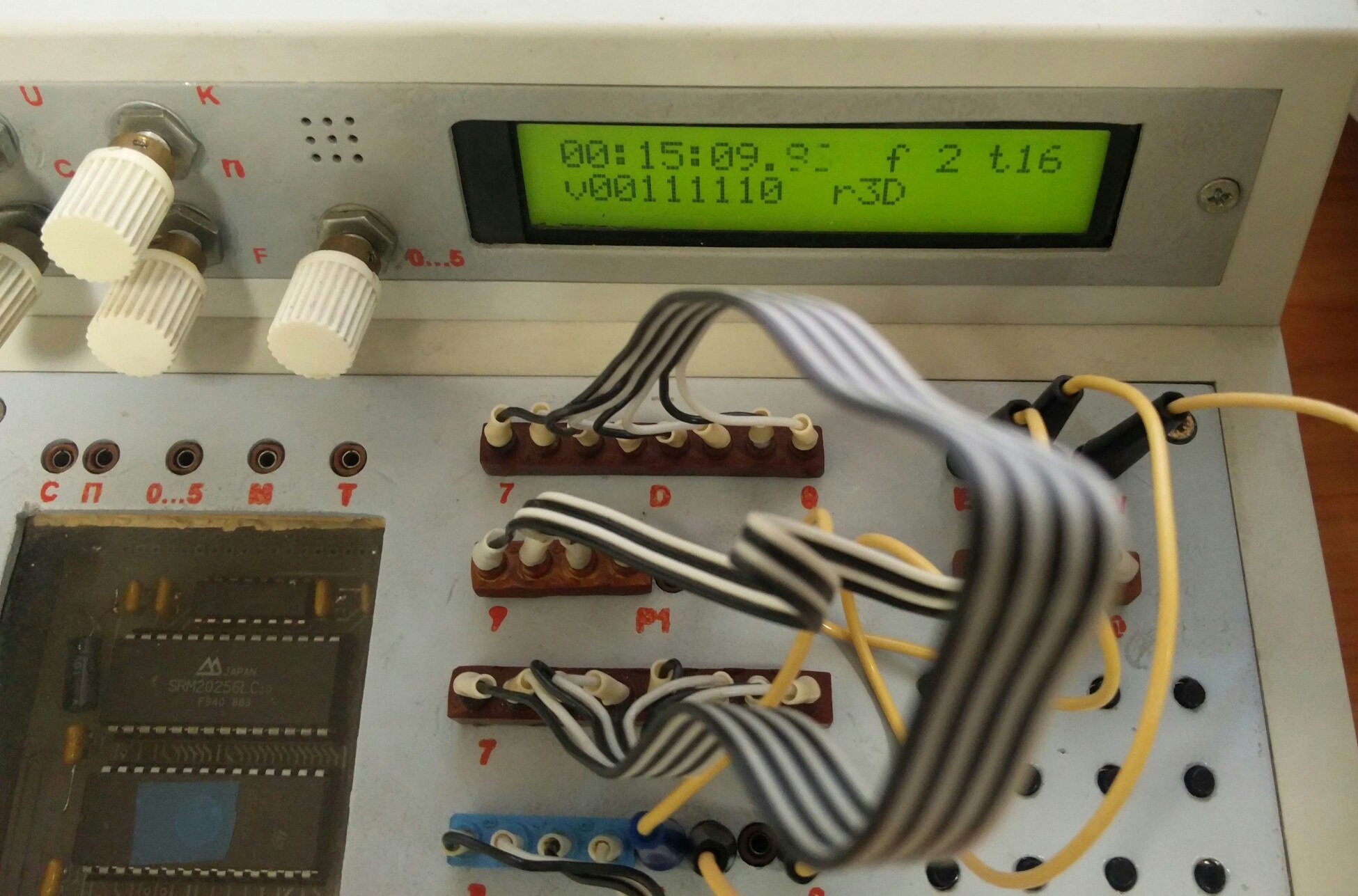
Результат работы программы (f – СС из которой переводим, t – СС в которую переводим, v – переводимое значение, r – переведенное значение):



*Рис. 7. Результат перевода числа 5510 в 678*



*Рис. 8. Результат перевода числа F516 в 111101012*



*Рис. 9. Результат перевода числа 001111102 в 3D16*

1. **Вывод**

В ходе работы был реализован преобразователь кодов, управляемый клавиатурой. Параллельно с преобразователем работает модуль индикации и модуль секундомера.

Согласно техническому заданию, программа переводит числа между СС с основаниями 2, 8, 10, 16. Однако программу легко изменить таким образом, чтобы поддерживались все операционные системы с основаниями 2-16 (ограничены только количеством кнопок на клавиатуре). Для этого всего лишь необходимо изменить набор параметров со стандартных на FROMSYS={2, 3, 4 … 15, 16}, TOSYS={2, 3, 4 … 15, 16}, MAX={8, 5, 4 … 3, 2}, DEF={128, 243, 64 … 225, 16}.

Также я убедился в том, что для корректной работы программ, использующих клавиатуру обязательно нужно устанавливать защиту от дребезга. Защита, выполненная в виде программной задержки на шестнадцатибитном счетчике, хорошо показала себя на практике. Также полезной опцией такого вида защиты от дребезга, является возможность собственной настройки длительности программной задержки.