РАДИОЛЮБИТЕЛЮ О МИКРОПРОЦЕССОРАХ И МИКРО-ЭВМ

ЗНАКОМСТВО ПРОДОЛЖАЕТСЯ

Г. ЗЕЛЕНКО, В. ПАНОВ, С. ПОПОВ

Знакомство с программмированием мы продолжим с разбора задачи, предложенной читателям в качестве домашнего задания в предыдущем номере журнала. Она заключалась в том, чтобы написать программу, обнуляющую область памяти, начиная с ячейки **0100H** по **02FFH** включительно.

Прежде всего напомним, что **0100** — это адрес ячейки, а **H** указывает на то, что записан он в шестнадцатиричном виде.

В нашей программе независимо от алгоритма, выбранного для решения задачи, обязательно будет несколько команд, с помощью которых в нее вводят исходные данные. Например, в любом варианте программы для обнуления хотя бы одной ячейки памяти нам понадобится байт (константа), во всех разрядах которого будут записаны нули, т. е. **00H**. Для задания такой константы воспользуемся командой **MVI M,00H**. Выполняя эту команду, микропроцессор запишет в ячейку памяти, адресуемую по содержимому регистровой пары **HL**, содержимое второго байта команды — в нашем случае **00H**. Естественно, таким образом в память можно записать и любой другой восьмиразрядный код, определяемый вторым байтом команды.

Еще мы должны будем указать микропроцессору (в программе) адрес ячейки памяти, с которой начинается область, подлежащая стиранию (ведь обнулить память это и означает стереть записанную в ней ранее информацию). Для этого в самом начале нашей программы по команде **LXI H,0100H** запишем в регистровую пару **HL** адрес этой ячейки — **0100H**. Теперь можно было бы записать в ячейку с этим адресом нуль (т. е. заслать в нее подготовленную нами константу), затем с помощью команды **INX H** прибавить к адресу этой ячейки единицу и, получив адрес следующей ячейки, заслать в нее нашу константу и так далее вплоть до ячейки с адресом **02FFH**.

Если пойти таким путем, то наша программа, не считая нескольких начальных и конечных команд, состояла бы из пар однотипных команд — однобайтовой команды прибавления единицы к текущему адресу и двухбайтовой команды засылки по этому адресу константы **00H**, повторенных по 512 раз, так как именно столько ячеек занимает область памяти с ячейки **0100H** по **02FFH**. Совершенно очевидно, что решение поставленной задачи «в лоб» привело бы нас к не очень экономному расходу бумаги, а самое главное — к прямо-таки варварскому использованию ячеек памяти. Для нашей программы пришлось бы отвести более 1500 ячеек памяти микропроцессора.

Давайте разберем более экономичный вариант программы. Распечатка программы этого варианта приведена на рис. 1. Он, конечно, далеко не единственный, но, с нашей точки зрения, достаточно разумный.

Так как нам предстоит обнуление последовательности ячеек, то организуем циклическую работу программы. В каждом цикле будем обнулять одну ячейку и затем подготавливать адрес очередной ячейки памяти для ее обнуления в следующем цикле. Для этого в цикле необходимо выполнять команду **INX H**, увеличивающую каждый раз на 1 содержимое регистров **HL**.

Работа программы должна прекратиться после обнуления последней ячейки памяти заданной области. В нашем случае это будет ячейка с адресом **02FFH**, загружаемым в регистровую пару **DE** по команде **LXI D,02FFH**. В ходе выполнения каждого цикла программы необходимо следить, чтобы постоянно увеличивающееся значение адреса в регистровой паре **HL** не превысило значения конечного адреса области памяти в регистровой паре **DE**. Для этого в каждом цикле программы необходимо сравнивать старшие байты адресов текущей и конечной ячеек памяти, т. е. коды в регистрах **H** и **D**. Это можно сделать вычитанием первого кода из второго (команды **MOV A,H, SUB D** и **JNZ НАЧАЛО**). При равенстве этих кодов проводится аналогичная проверка на равенство значений младших байтов адресов текущей и конечной ячеек обнуляемой области памяти. Достижение такого равенства означает, что в **HL** уже находится адрес конечной ячейки памяти, поэтому необходимо обнулить эту ячейку (предпо-

Рис. 1

```
*************************************************************************
! АДР.! КОД  ! МЕТКА ! МНЕМ. ! ОПЕРАНД ! КОММЕНТАРИЙ
!  1  !  2   !   3   !   4   !    5    !        6
*************************************************************************
 1000  210001          LXI     H,0100H ; ЗАГРУЗКА АДРЕСА НАЧАЛА
                                       ; ОБНУЛЯЕМОЙ ЗОНЫ ПАМЯТИ.
 1003  11FF02          LXI     D,02FFH ; ЗАГРУЗКА АДРЕСА КОНЦА
                                       ; ОБНУЛЯЕМОЙ ЗОНЫ ПАМЯТИ.
 1006  3600    НАЧАЛО: MVI     M,00H   ; ОБНУЛЕНИЕ ЯЧЕЙКИ ПАМЯТИ.
 1008  23              INX     H       ; ПОДГОТОВКА ОЧЕРЕДНОГО
                                       ; АДРЕСА ОБНУЛЯЕМОЙ ЯЧЕЙКИ.
 1009  7C              MOV     A,H     ; СРАВНЕНИЕ СТАРШИХ БАЙТОВ
 100A  92              SUB     D       ; АДРЕСОВ ТЕКУЩЕЙ И КОНЕЧНОЙ
 100B  C20610          JNZ     НАЧАЛО  ; ЯЧЕЕК ПАМЯТИ.
 100E  7B              MOV     A,E     ; СРАВНЕНИЕ МЛАДШИХ БАЙТОВ
 100F  95              SUB     L       ; АДРЕСОВ ТЕКУЩЕЙ И КОНЕЧНОЙ
 1010  C20610          JNZ     НАЧАЛО  ; ЯЧЕЕК ПАМЯТИ.
 1013  3600          MVI     M,00H   ; ОБНУЛЕНИЕ ПОСЛЕДНЕЙ ЯЧЕЙКИ.
 1015  76              HLT             ; ОКОНЧАНИЕ ПРОГРАММЫ.
```

Продолжение. Начало см. в «Радио», 1982, № 9—11.

следняя команда программы) и прекратить выполнение программы.

У начинающих программистов обычно вызывает трудности использование команд условной передачи управления, например, при сравнении значений двух байтов. Напомним, что перед командой условной передачи управления всегда располагается команда, воздействующая на соответствующий бит регистра признаков F. В предыдущем примере для сравнения двух байтов в качестве такой команды использовалась команда вычитания, а передачи управления осуществлялась по команде **JNZ ADR**, контролирующей состояние бита **Z**. Вместо команд вычитания можно использовать и другие команды, например команды сравнения **CPI D8** или **CPI M**. Действия, оказываемые этими командами на биты регистра F, зависят от результата операции **A—D8** или **A—M**, но, в отличие от других команд, они не изменяют предшествующего содержимого аккумулятора.

Команды **JC ADR** и **JNC ADR** осуществляют передачу управления соответственно в случаях, когда **M** (или **D8**) $>$ **A** и **M** (или **D8**) $\leqslant$ **A**.

Так, например, последовательность команд

CPI 10D
JNC ADR

осуществляет передачу управления, если **A** $\geqslant$ **10 D**, а последовательность команд

CPI 10D
JC ADR,

если **A** $<$ **10D**.

Предположим, что содержимое аккумулятора **A** к моменту выполнения этих команд будет равно **10D**. Тогда в первом случае будет осуществлена передача управления на команду, расположенную в ячейке с адресом **ADR**, а во втором случае передача управления не произойдет и будет выполняться следующая по порядку команда.

Следующий наш пример посвящен программной реализации такого распространенного цифрового элемента, как дешифратор для семисегментного индикатора.

Предположим, что имеется входной порт 0, к которому подключены четыре тумблера, образующие тумблерный регистр. Оператор может набирать на этом тумблерном регистре различные кодовые комбинации. Имеется также выходной порт 1, к которому подключен семисегментный индикатор. Программа должна считывать информации с тумблерного регистра и отображать соответствующую десятичную цифру на индикаторе.

В табл. 1 дано соответствие между кодовыми комбинациями, набираемыми на тумблерном регистре, байтами на выходе порта 1 и десятичной цифрой на семисегментном индикаторе. Байт на выходе порта 1 будем называть семисегментным кодом. На рис. 2 условно показано подключение индикатора к порту 1.

Таблица 1

Кодовая комбинация	Семисегментный код	Десятичная цифра
0000	3F	0
0001	06	1
0010	5B	2
0011	4F	3
0100	66	4
0101	6D	5
0110	7D	6
0111	07	7
1000	7F	8
1001	6F	9
1010	Запрещенные комбинации	—
...		...
1111		—

На рис. 3 приведена распечатка программы.

Поставленная задача наиболее просто может быть решена, если мы воспользуемся следующим приемом. Поместим в десяти последовательно расположенных вслед за программой ячейках памяти семисегментные коды, приведенные в табл. 1. На то, что в ячейках памяти находятся не коды команд, а используемые в программе константы или операнды, указывают символы **DB** (сокращение от английского выражения «определи байт»), помещаемые в поле 4 распечатки. В поле 5 напротив этих символов заносятся числа, которые должны быть записаны в память до начала выполнения программы. Адресу первой из этих ячеек присвоим метку **ТАБЛ.** Начало области памяти или отдельные ячейки памяти, содержащие различные операнды, используемые в программах, могут быть отмечены метками точно так же, как и команды программы.

Наша программа будет «работать» следующим образом. Сначала в аккумулятор введется содержимое тумблерного регистра. Так как четыре старших разряда порта не используются, то они «маскируются» выполнением команды логического умножения содержимого аккумулятора на операнд, у которого в старших четырех разрядах записаны нули, а в младших четырех разрядах — единицы. Далее проводится проверка на допустимость числа, считанного с тумблерного регистра (не превышает ли он девяти). Обратите внимание, что операнд команды **CPI 10D** на единицу больше, чем допустимое вводимое число. Подумайте почему. Если код числа недопустим, то вновь производится считывание содержимого тумблерного регистра. Каждому допустимому числу ставится в соответствие семисегментный код. Для этого в регистровую пару **HL** помещается адрес метки **ТАБЛ**, обнуляется содержимое регистровой пары **DE** и затем двоичный код из аккумулятора пересылается в регистр **E**. Если теперь сложить содержимое **HL** и **DE**, то **HL** будет содержать адрес ячейки памяти, в которой хранится соответствующий семисегментный код. Этот код по команде **MOV A,M** пересылается в порт 1, к которому подключен семисегментный индикатор. После этого программа вновь возвращается на считывание содержимого тумблерного регистра.

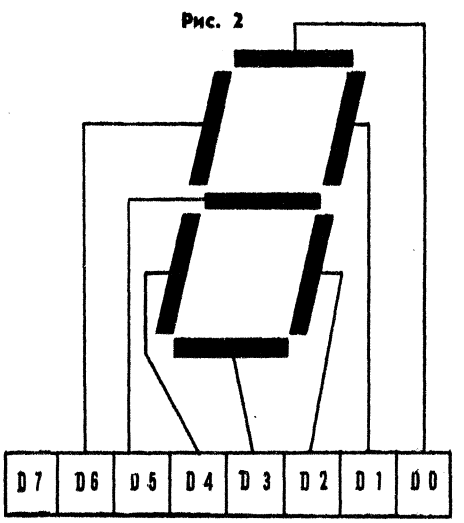
Описанный нами прием использования таблицы, хранящейся в памяти, может быть применен в самых различных случаях дешифрации и преобразования кодов.

Таблица 2

```
0100 DB 00 E6 0F FE 0A D2 00 01 21 17 01 11 00 00 5F
0110 19 7E D3 01 C3 00 01 3F 06 5B 4F 66 6D 7D 07 7F
0120 6F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

В табл. 2 приведена рассмотренная нами программа в компактной форме в виде содержимого области памяти, хранящей нашу программу. Каждая строка таблицы (в нашем примере таких строк только три) начинается с четырехразрядного шестнадцатиричного числа. Это число — адрес ячейки памяти, в которой записан первый из шестнадцати последовательно расположенных в памяти байтов, представленных далее в строке двухразрядными шестнадцатиричными числами. Распечатки содержимого памяти в таком виде мы будем часто приводить для экономии места в журнале. По такой таблице можно восстановить текст программы, если воспользоваться таблицей кодов команд микропроцессора и при этом точно

Рис. 2



знать, в каких ячейках записаны команды программы, а в каких — константы или промежуточные данные. Иначе промежуточные данные будут «расшифрованы» как команды, что внесет путаницу.

Микропроцессорные устройства и микро-ЭВМ наиболее часто работают в так называемом «режиме реального времени». Этот режим характерен тем, что события во внешнем по отношению к микропроцессорному устройству мире происходят в различные непредсказуемые заранее моменты времени. Микропроцессор должен своевременно реагировать на эти события независимо от того, занят ли он в данный момент какими-либо другими действиями или нет. Для этого имеется возможность прерывания работы текущей программы по специальным сигналам от внешних устройств или датчиков. При поступлении запроса прерывания микропроцессор переходит к выполнению подпрограммы обработки прерывания, то есть к действию, являющемуся реакцией на внешнее событие. При появлении запроса прерывания микропроцессор после выполнения очередной команды текущей программы считывает не как обычно код операции следующей команды из памяти, а код команды вызова подпрограммы, формируемый на шинах данных специальным блоком — контроллером прерываний. Что собой представляет контроллер прерываний, будет описано в последующих статьях.

Обычно в качестве команд вызова подпрограмм используют однобайтовые команды **RST 0 — RST 7**. В зависимости от номера команды ее выполнение ведет к передаче управления на одну из ячеек в начальной области памяти (смотри систему команд микропроцессора). Именно с команды в этой ячейке и должна начинаться подпрограмма обслуживания прерывания.

Чтобы уяснить работу механизма прерываний, рассмотрим конкретный пример. Представим себе высококачественный магнитофон, в котором управление всей его работой, а также автоматическая стабилизация натяжения ленты и индикация числа оборотов производятся специализированным устройством. В режиме воспроизведения устройство должно «заниматься» определением степени натяжения ленты, расчетом управляющего воздействия, выдачей его на исполнительное устройство и в то же время подсчитывать число импульсов от датчика оборотов и отображать его на соответствующем индикаторе.

Таким образом, основная программа в этом режиме — это замкнутый цикл считывания состояния датчика натяжения ленты, расчет управляющего воздействия и выдача его к исполнительному механизму. Так как сигналы от датчика оборотов приходят довольно редко и моменты их появления не связаны с работой основной программы, то целесообразно использовать их в качестве источников запросов прерывания. Предположим, что по каждому запросу прерывания на шине данных контроллер прерывания формирует команду **RST 7**. Следовательно, первая команда обслуживания запроса прерывания должна располагаться в ячейке памяти с адресом **0038H**. Эта программа начинается следующими командами:

```
PUSH PSW
PUSH B
PUSH D
PUSH H
```

Четыре первые позволяют сохранить в стеке содержимое всех регистров микропроцессора, для того чтобы после возврата к основной программе можно было восстановить их содержимое. Дальнейший текст подпрограммы здесь не приводится, так как в данном случае нам важны ее фрагменты, специфичные для обработки прерывания. В конце подпрограммы выполняются следующие команды:

```
POP H
POP D
POP B
POP PSW
EI
RET
```

```
**********************************************************************
 АДР.! КОД  ! МЕТКА ! МНЕМ. ! ОПЕРАНД !          КОММЕНТАРИЙ
  1  !  2   !  3   !  4    !    5    !               6
**********************************************************************
0100  DB00    НАЧАЛО: IN      00H     ;ВВОД СОСТОЯНИЯ ТУМБЛЕРОВ.
0102  E60F            ANI     0FH     ;МАСКИРОВАНИЕ НЕИСПОЛЬЗУЕМЫХ
                                      ;РАЗРЯДОВ.
0104  FE0A            CPI     10D     ;ПРОВЕРКА НА ДОПУСТИМОСТЬ
                                      ;ВВОДИМОГО КОДА.
0106  D20001          JNC     НАЧАЛО  ;ЕСЛИ ВВЕДЕННОЕ ЧИСЛО БОЛЬШЕ 9,
                                      ;ТО ПЕРЕЙТИ НА "НАЧАЛО:".
0109  211701          LXI     H,ТАБЛ  ;ЗАГРУЗИТЬ В HL НАЧАЛЬНЫЙ
                                      ;АДРЕС ТАБЛИЦЫ КОДОВ.
010C  110000          LXI     D,0000  ;ОБНУЛИТЬ DE.
010F  5F              MOV     E,A     ;ЗАГРУЗИТЬ В E ВВЕДЕННОЕ ЧИСЛО,
                                      ;ЗНАЧЕНИЕ КОТОРОГО ИСПОЛЬ-
                                      ;ЗУЕТСЯ КАК СМЕЩЕНИЕ.
0110  19              DAD     D       ;ВЫЧИСЛИТЬ И ПОМЕСТИТЬ В HL
                                      ;АДРЕС СЕМИСЕГМЕНТНОГО КОДА.
0111  7E              MOV     A,M     ;ЗАГРУЗИТЬ В A СЕМИСЕГ. КОД.
0112  D301            OUT     01H     ;ВЫВЕСТИ КОД НА ИНДИКАТОР.
0114  C30001          JMP     НАЧАЛО  ;ПЕРЕЙТИ НА "НАЧАЛО:".
                                      ;ТАБЛИЦА СЕМИСЕГМ.КОДОВ
0117  3F        ТАБЛ: DB      3FH     ;КОД ЦИФРЫ 0
0118  06              DB      06H     ;КОД ЦИФРЫ 1
0119  5B              DB      5BH     ;КОД ЦИФРЫ 2
011A  4F              DB      4FH     ;КОД ЦИФРЫ 3
011B  66              DB      66H     ;КОД ЦИФРЫ 4
011C  6D              DB      6DH     ;КОД ЦИФРЫ 5
011D  7D              DB      7DH     ;КОД ЦИФРЫ 6
011E  07              DB      07H     ;КОД ЦИФРЫ 7
011F  7F              DB      7FH     ;КОД ЦИФРЫ 8
0120  6F              DB      6FH     ;КОД ЦИФРЫ 9
```

Рис. 3

С помощью четырех команд чтения из стека восстанавливается содержимое регистров микропроцессора, а затем выполняется команда разрешения прерывания **EI**. Последнее необходимо, так как после возникновения прерывания в микропроцессоре всегда автоматически запрещается прием запросов прерываний. Последняя команда **RET** производит возврат в основную программу: в данном случае программу стабилизации натяжения ленты. Возврат происходит в то место этой программы и с тем состоянием внутренних регистров, которые были до момента возникновения прерывания. Попутно заметим, что микропроцессорное устройство, встроенное в магнитофон, позволяет не только улучшить его качественные и эксплуатационные характеристики, но и значительно расширить его возможности.

Как вы уже поняли, перевод текста программы в машинные коды является очень кропотливой работой и порождает много ошибок. Этот процесс может быть автоматизирован с помощью специальной сложной программы — транслятора, транслирующей (переводящей) исходные тексты программ в машинные коды. Такая программа называется ассемблером, и поэтому тексты наших

программ, записанные в полях 3, 4, 5 и 6 распечаток программ, являются текстами программ на языке ассемблера (ассемблерными текстами). В радиолюбительской практике на первых порах придется мириться с ручной трансляцией ассемблерного текста. При этом можно создавать программы объемом до нескольких сотен команд. Для радиолюбительских конструкций программы такой сложности могут оказаться вполне приемлемыми.

Конечно, впервые писать программы весьма сложно, но мы при описании модулей микро-ЭВМ будем приводить готовые распечатки программ. Разбирая эти программы, вы можете получить некоторые навыки программирования.

Готовыми стандартными программами пользуются и профессиональные программисты. Обычно алгоритмы, реализующие различные математические операции: умножение, деление, вычисления тригонометрических и логарифмических функций, решение систем уравнений и др., — оформляются в виде набора стандартных подпрограмм.

Для еще большего облегчения процесса программирования были разработаны языки высокого уровня: БЕЙСИК, ПАСКАЛЬ, ФОРТРАН и др. Программист, пишущий программы на языке высокого уровня, может вообще не знать устройство микро-ЭВМ и ее систему команд. Поэтому при написании программы программист может использовать выражения естественного языка (если..., то; до тех пор, пока...; исполнить...). Однако объем занимаемой памяти (он во многом определяет стоимость устройств) у программ, написанных на языке высокого уровня, существенно больше, чем у программ, написанных на ассемблере. К тому же программы, написанные на языке высокого уровня, работают медленнее. Поэтому программы для специализированных устройств, когда важны стоимость и скорость реализации алгоритма, пишут все же в большинстве случаев на ассемблере.

В одной статье трудно рассмотреть все многообразие методов и приемов программирования. Мы ставили перед собой задачу дать общее начальное представление о программировании на языке ассемблера и в машинных кодах. Рассмотренные примеры показывают, что микропроцессор, работая по определенной программе, может выполнять функции самых различных цифровых устройств.

Основываясь на приведенных в статье примерах, читатель может писать небольшие программы. Для более глубокого изучения программирования для микропроцессоров можно обратиться к литературе, указанной в конце статьи.

ЛИТЕРАТУРА

Микро-ЭВМ.— М.: Энергоиздат, 1982.

ДИНАМИЧЕСКИЙ ФИЛЬТР «МАЯК»

И. ИЗАКСОН, А. НИКОЛАЕНКО, В. СМИРНОВ

Как известно, благодаря психоакустическому эффекту составляющие шума, на частотах которых присутствуют полезные компоненты сигнала с более высоким уровнем, при звуковоспроизведении не прослушиваются (маскируются). Остальные составляющие шума хорошо слышны и мешают восприятию звуковой программы. Ослабить их можно изменением полосы пропускания канала звукопередачи таким образом, чтобы через него свободно проходили только полезные составляющие сигнала, а незамаскированные компоненты шума оставались за пределами полосы. Этот принцип и положен в основу работы так называемых динамических шумопонижающих фильтров (далее — просто динамических фильтров).

Основное достоинство динамического фильтра — возможность снижения шума не только канала звукопередачи, но и самой звуковой программы (ее реставрация), недостаток — изменение динамического диапазона составляющих сигнала, уровень которых ниже порога шумопонижения. Недостатком шумоподавителей этого типа является и эффект модуляции шума (впрочем, в той или иной степени он присущ любой системе шумопонижения). Этот эффект проявляется в слышимых колебаниях уровня шума при изменении уровня составляющих полезного сигнала. В динамических фильтрах эффект модуляции шума обусловлен тем, что несущие максимальную энергию среднечастотные составляющие звуковой программы влияют на полосу пропускания: например, увеличение их уровня при отсутствии маскирующих высокочастотных составляющих ведет к чрезмерному расширению полосы пропускания канала звукопередачи (она становится шире спектра полезного сигнала), в результате чего высокочастотный шум на выходе устройства возрастает; при снижении уровня среднечастотных составляющих полоса пропускания сужается и этот шум уменьшается.

От указанных недостатков в значительной мере свободен динамический фильтр, получивший условное название «Маяк»*. Благодаря повышению точности управления полосой пропускания в зависимости от спектра звуковой программы этот фильтр обеспечивает эффективное понижение шума при минимальных искажениях динамики звуковой программы и снижении эффекта модуляции шума.

Структурная схема динамического фильтра «Маяк» представлена на рис. 1. Он состоит из управляемого (*Z2*) и неуправляемого (*Z1*) фильтров нижних частот (ФНЧ), алгебраического сумматора *A1*, весового фильтра *Z3*, ограничителя минимума *U1*, дифференциатора *U2* и амплитудного детектора *U3*.

Управляемый ФНЧ *Z2* с регулируемой частотой среза предназначен для изменения полосы пропускания звуковоспроизводящего тракта. Его вход и выход являются входом и выходом устройства в целом. В исходном состоянии (при отсутствии сигнала на входе) частота среза этого ФНЧ равна 1,5 кГц, в процессе обработки сигнала она может перестраиваться вплоть до верхней граничной частоты рабочего диапазона.

Как видно их схемы, обрабатываемый сигнал поступает одновременно и на вход неуправляемого ФНЧ *Z1*. Частота среза этого фильтра равна верхней граничной частоте рабочего диапазона, а АЧХ и ФЧХ такие же, как и у управляемого ФНЧ *Z2* на этой частоте. Сигналы с выходов ФНЧ *Z1* и *Z2* поступают в алгебраический сумматор *A1*. В результате вычитания из широкополосного выходного сигнала неуправляемого ФНЧ *Z1* составляющих сигнала, прошедших через управляемый ФНЧ *Z2*, в сумматоре *A1* формируется сигнал, составляющие которого располагаются выше частоты среза управляемого ФНЧ *Z2*. Через весовой фильтр *Z3*, дополнительно ослабляющий влияние низкочастотных составляющих, этот сигнал поступает в ограничитель минимума *U1* (он служит для установки порога шумопонижения) и далее на вход дифференциатора *U2*, изменяющего уровень сигнала управления, превышающего порог срабатывания, пропорционально частоте. Амплитудный детектор *U3* преобразует переменное напряжение управляющего сигнала в постоянное с требуемыми временами установления и восстановления. Управляющий сигнал с выхода амплитудного детектора поступает на управляющий вход ФНЧ *Z2*. Напря-

* Авторское свидетельство СССР № 734868 (бюллетень «Изобретения, открытия...», 1980, № 18). Запатентован в США (патент № 4.207.543 от 10.06.1980 г.) и Франции (патент № 2.435.156 от 20.02.1981 г.).