РАДИОЛЮБИТЕЛЮ О МИКРОПРОЦЕССОРАХ И МИКРО-ЭВМ

ЗНАКОМСТВО С ПРОГРАММИРОВАНИЕМ

Г. ЗЕЛЕНКО, В. ПАНОВ, С. ПОПОВ

Микропроцессорные устройства и микро-ЭВМ выполняют свои функции в соответствии с программами, составленными и отлаженными разработчиками и записанными затем в память машины.

Каким же образом составляют программы для микро-ЭВМ? Давайте рассмотрим подробно простой пример. Требуется составить программу работы специализированного микропроцессорного устройства. Пусть в таком устройстве имеются два порта — один порт ввода *D2* и один порт вывода *D4*. К порту ввода *D2* по линии, связанной с его младшим разрядом, подключена кнопка *S1*, а к порту вывода *D4* по линиями, связанным с его двумя младшими разрядами, подключены два светодиода. Функциональная схема части этого устройства приведена на рис. 1. Дешифраторы *D1* и *D3* формируют сигналы $\overline{\mathbf{BM1}}$ и $\overline{\mathbf{BM2}}$, служащие для активизации соответствующих портов. При этом дешифраторы включены таким образом, что сигналы $\overline{\mathbf{BM1}}$ и $\overline{\mathbf{BM2}}$ появляются только тогда, когда на младших восьми разрядах шины адресов возникают коды 01 и 02 соответственно. В таком случае говорят, что порт *D2* включен как устройство с номером 1, а порт *D4* — с номером 2. Далее порт *D2* будем называть портом 1, а порт *D4* — портом 2.

Допустим, что от устройства требуется выполнение следующей задачи. В исходном состоянии, пока кнопка *S1* не нажата, светодиод *V1* должен светиться, а светодиод *V2* нет. Если кратковременно нажать на кнопку, то светодиод *V1* должен на 0,25 с погаснуть, а светодиод *V2* загореться. После этого в течение 0,5 с устройство не должно реагировать на нажатие кнопки.

Конечно, использовать микро-ЭВМ для решения такой простой задачи нецелесообразно, но для первого шага и знакомства с принципами составления программ именно такая задача нам и нужна.

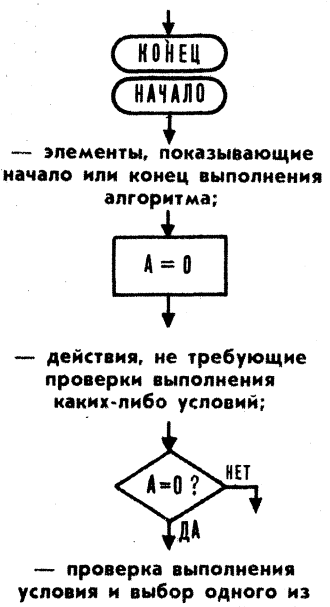
Теперь немного отвлечемся. Надо заметить, что аккуратность — полезное качество в любом деле, а в программировании оно просто необходимо. Поэтому приготовим для справок материалы предыдущих статей, запасемся карандашом, бумагой (и, конечно, терпением) и двинемся дальше.

Итак, наша программа должна начинаться с записи в порт 2 комбинации битов 00000001. Именно при этом условии *V1* горит, а *V2* погашен. Затем по команде ввода содержимое порта 1 пересылается в аккумулятор и анализируется, не появился ли ноль в младшем разряде, что может быть только при нажатии на кнопку. Если кнопка не нажата, то операция повторяется. В противном случае в порт 2 посылается комбинация 00000010 (светодиод *V2* горит, *V1* погашен). После этого микропроцессор в течение 0,25 с не должен производить никаких операций с портами, а затем после полусекундной задержки вновь перейти к началу нашей программы. Эти задержки можно будет выполнить программно с помощью специальной подпрограммы.

Занесем все перечисленные действия в таблицу и пронумеруем каждый шаг. Таблица — это, по существу, алгоритм решения поставленной задачи.

Представление алгоритма в виде таблицы не очень удобно. Более наглядны блок-схемы алгоритмов. В дальнейшем мы и будем их использовать.

Блок-схемы алгоритмов могут состоять всего из четырех основных элементов. Графически они выглядят так:



— элементы, показывающие начало или конец выполнения алгоритма;

— действия, не требующие проверки выполнения каких-либо условий;

— проверка выполнения условия и выбор одного из двух возможных действий в зависимости от ее результатов;

— ввод информации от внешних устройств или вывод во внешние устройства.

Шаг	Выполняемые действия
1	Записать в аккумулятор код 00000001 и переслать его в порт 2
2	Выполнить программу, реализующую задержку в 0,5 с
3	Переслать содержимое порта ввода 1 в аккумулятор
4	Проверить состояние разряда А (0) аккумулятора
5	Если А (О) равно 0, то перейти к шагу 3, в противном случае перейти к шагу 6
6	Записать в аккумулятор код 00000010 и переслать его в порт 2
7	Выполнить программу, реализующую задержку в 0,25 с
8	Перейти к шагу 1

Продолжение. Начало см. в «Радио», 1982, №№ 9, 10.

Блок-схема нашего алгоритма приведена на рис. 2. Сплошными линиями со стрелками показан ход его выполнения. Справа от блок-схемы мы поместили программу, под управлением которой микропроцессор выполняет алгоритм. Штриховые линии со стрелками лишь указывают на соответствие элементов блок-схемы алгоритма и команд программы. Эти линии не являются принадлежностью блок-схемы и включены в данном случае только для наглядности.

Теперь мы вновь обратимся к таблице и увидим, что при выполнении алгоритма нам неоднократно придется возвращаться к шагам 1 и 3. Поэтому эти шаги должны быть особо отмечены с помощью меток — произвольно выбираемых имен. На рис. 2 в прямоугольниках из штриховых линий в соответствующих местах блок-схемы алгоритма показаны эти метки. Они обычно содержат знак двоеточия после своего имени, а имена меток выбираются таким образом, чтобы было ясно их назначение (в рассматриваемом примере мы будем использовать метки **НАЧАЛО, ВВОД, ВРЕМЯ**).

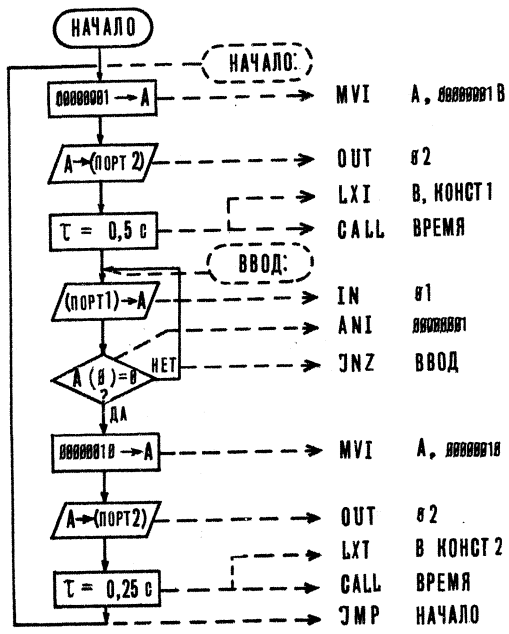


Рис. 1

Теперь, когда блок-схема алгоритма составлена, можно перейти к написанию программы. Трудности при написании программы в некотором смысле сродни трудностям при разработке аппаратуры. От разработчика программы также требуется знание некоторых приемов программирования и типовых программных решений. Пользуясь здравым смыслом, комбинируя стандартные и свои оригинальные решения, он должен получать программы с заданными свойствами. Обучиться программированию на первоначальном этапе можно, разбирая программы, написанные другими программистами и составляя собственные программы.

Но вернемся к нашему алгоритму. Пользуясь описанием системы команд и таблицей 2 предыдущей статьи, нетрудно понять, что привести устройство в начальное состояние можно с помощью последовательности команд: команды загрузки операнда в аккумулятор **MVI A,00000001** и команды вывода байта данных из аккумулятора в порт 2 — **OUT 02**. Затем следуют две команды, необходимые для организации временной задержки (их мы разберем несколько позже), и далее команда опроса состояния кнопки *S1* (т. е. значения младшего разряда порта 1). С помощью команды ввода **IN 01** переписываем содержимое порта в аккумулятор и проверяем состояние его младшего бита **A** (0). Для этого воспользуемся командой логического умножения содержимого аккумулятора на непосредственный операнд (команда **ANI 00000001**), содержащий 1 только в младшем разряде. В результате выполнения этой команды в аккумулятор запишется код, семь старших разрядов которого будут равны 0. Значение нулевого разряда аккумулятора **A** (0) будет зависеть от значения младшего разряда кода, прочитанного из порта 1, т. е. от того, была ли нажата кнопка в момент действия команды **IN 01**.

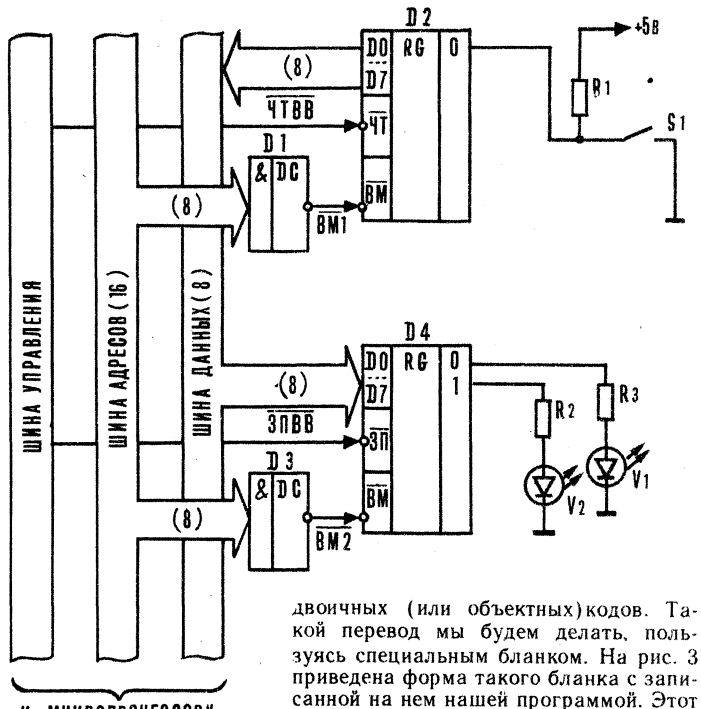


Рис. 2

Команда логического умножения воздействует на все биты признаков регистра **F**. Нас в данном случае интересует состояние бита **Z** — признака нуля, который устанавливается в 1 при нулевом содержимом аккумулятора, что в данном случае соответствует нажатой кнопке. Таким образом, команда **JNZ ВВОД** будет передавать управление команде **IN 01**, помеченной меткой **ВВОД**, до тех пор, пока признак **Z** остается равным нулю, нажатие на кнопку приведет к установке признака **Z** в 1 и выполнению последующих команд программы.

При написании программы мы пользовались мнемоническими обозначениями команд микропроцессора и символическими именами для меток. Конечно, программа в таком виде не может быть непосредственно выполнена на микро-ЭВМ. Для этого она должна быть переведена на машинный язык — язык двоичных (или объектных) кодов. Такой перевод мы будем делать, пользуясь специальным бланком. На рис. 3 приведена форма такого бланка с записанной на нем нашей программой. Этот документ программисты часто называют распечаткой программы. Каждая строка бланка разделена на 6 полей. При написании программы первоначально используют поля с 3-го по 6-е. В поле 4 записывают мнемоническое обозначение выполняемой команды. В поле 5 — необходимый операнд (адрес перехода, номер порта, наименование регистра и т. д.). Операнд может быть задан непосредственно в виде конкретного числа или неявно, т. е. ему может быть присвоено символическое имя, а конкретное значение, соответствующее этому имени, определяется в дальнейшем при переводе текста программы в машинные коды. Когда операнд задается непосредственно, то после него проставляется буква **D**, **H** или **B**, если число представлено в десятичной, шестнадцатиричной или двоичной форме соответственно. Поле 6 отведено для комментариев (пояснения действий, выполняе-

```
*************************************************************************
! АДР.! КОД  ! МЕТКА ! МНЕМ. ! ОПЕРАНД !          КОММЕНТАРИЙ          !
!  1  !  2   !   3   !   4   !    5    !            6                   !
*************************************************************************
 0000  3E01    НАЧАЛО: MVI     A,01H    ; 00000001 --> A.
 0002  D302            OUT     02H      ; ЗАЖЕЧЬ V1, ПОГАСИТЬ V2.
 0004  0170CB          LXI     B,КОНСТ1 ; ЗАДАТЬ ВЕЛИЧ. ЗАДЕРЖКИ 0,5 С.
 0007  CD1E00          CALL    ВРЕМЯ    ; ВЫЗОВ ПОДПРОГРАММЫ ЗАДЕРЖКИ.
 0009  DB01      ВВОД: IN      01H      ; ВВОД БАЙТА ИЗ ПОРТА 1.
 000C  E601            ANI     01H      ; МАСКИРОВАНИЕ НЕИСПОЛЬЗУЕМЫХ
                                        ; РАЗРЯДОВ.
 000E  C20A00          JNZ     ВВОД     ; ПОВТОРИТЬ ВВОД, ЕСЛИ КНОПКА
                                        ; НЕ НАЖАТА.
 0011  3E02            MVI     A,02H    ; 00000010 --> A.
 0013  D302            OUT     02H      ; ЗАЖЕЧЬ V2, ПОГАСИТЬ V1.
 0015  01B865          LXI     B,КОНСТ2 ; ЗАДАТЬ ВЕЛИЧ. ЗАДЕРЖКИ 0,25 С.
 0018  CD1E00          CALL    ВРЕМЯ    ; ВЫЗОВ ПОДПРОГРАММЫ ЗАДЕРЖКИ.
 001B  C30000          JMP     НАЧАЛО   ; ВОЗВРАТ НА "НАЧАЛО:".
                                        ; ПОДПРОГРАММА ЗАДЕРЖКИ.
 001E  0B        ВРЕМЯ: DCX     B        ; УМЕНЬШИТЬ НА 1 СОДЕРЖИМОЕ ВС.
 001F  78               MOV     A,B      ; ПЕРЕСЛАТЬ В А СОДЕРЖИМОЕ В.
 0020  B1               ORA     C        ; В И С РАВНЫ НУЛИ?
 0021  C21E00           JNZ     ВРЕМЯ    ; ЕСЛИ НЕТ, ТО ПОВТОРИТЬ ЦИКЛ.
 0024  C9               RET              ; ВОЗВРАТ В ОСНОВНУЮ ПРОГРАММУ.
                КОНСТ1  EQU     52080D   ; ПРИСВОЕНИЕ ЧИСЛОВЫХ ЗНАЧЕНИЙ
                КОНСТ2  EQU     26040D   ; СИМВОЛИЧЕСКИМ ОПЕРАНДАМ.
```

мых программой). Наличие этого поля необязательно, но если вы хотите, чтобы ваша программа была понятна другим, впрочем, как и вам самим по истечении некоторого времени после ее написания (детали быстро забываются!), комментарии должны присутствовать. Поле 3 заполняют в тех случаях, когда необходимо отметить начало фрагментов программ, к которым осуществляется переход из других частей программы по команде передачи управления и вызова подпрограмм.

Теперь о полях 1 и 2. Имея перед собой таблицу кодов команд микропроцессора КР580ИК80А, мы можем заполнить эти поля. Они всегда заполняются числами в шестнадцатиричной форме, и буква **H** в конце числа при этом опускается. Если мы при составлении программы задаем операнды неявно, то в начале или в конце текста программы определяем, чему же эти операнды равны на самом деле. Для этого в поле 3 записываем символическое имя операнда, в поле 4 — сокращенное английское слово **EQU** (равно), а в поле 5 — действительное значение операнда. В поле 2 записываем коды и соответствующие операнды команд. При этом символические имена заменяются их действительными значениями. При записи 16-разрядных (двухбайтовых) операндов и адресов в поле 5 мы придерживаемся естественной формы, т. е. слева записываем два старших, а правее — два младших разряда числа. При записи кодов операндов в поле 2 — порядок обратный, что объясняется порядком записи байтов 16-разрядных чисел в память микро-ЭВМ (см. «Радио», 1982, № 9).

Первое значение адреса в поле 1 (адрес первой команды программы) выбирается программистом в зависимости от свободного места в памяти микро-ЭВМ. Поле заполняют так, чтобы значения, указанные здесь, соответствовали адресам ячеек памяти, содержащих коды команд.

При рассмотрении системы команд вы узнали о том, что команды бывают одно-, двух- и трехбайтовые. Естественно, это нашло отражение и в формате поля 2: оно может содержать 1, 2 или 3 байта. Поэтому при расчете очередного адреса в поле 1, в зависимости от длины предшествующей команды, к предыдущему значению прибавляется 1, 2 или 3 (учтите, что здесь мы имеем дело с шестнадцатиричной арифметикой: например, 8+3=B, B+2=D, F+3=12 и так далее). Коды второго и третьего байта в командах передачи управления и вызова подпрограммы в поле 2 вносим в таблицу в последнюю очередь, после заполнения всех строк поля 1. При этом в поле 2 получаем запись программы в машинных кодах. Если теперь коды из поля 2 поместить в память микро-ЭВМ по адресам, указанным в поле 1, то при пуске микро-ЭВМ с начального адреса (в данном случае 0000) она начнет выполнять наш алгоритм.

Теперь настало время рассмотреть, как организуют в микро-ЭВМ временные задержки. Часто это осуществляется программно. В данном случае для этой цели используется подпрограмма **ВРЕМЯ**. В виде подпрограмм, как правило, оформляют часто повторяемые однотипные действия, встречающиеся в различных частях реализуемого алгоритма. Такой подход позволяет значительно сэкономить требуемый объем памяти и сделать основную программу более понятной. Вместо того, чтобы неоднократно писать в разных местах программы совершенно одинаковые последовательности команд, программист использует там только команды вызова подпрограмм, а нужную последовательность команд оформляет однократно в виде подпрограммы.

Обратиться к подпрограмме довольно просто: в основной программе для этого нужно подготовить исходные данные, поместив их в определенные регистры микропроцессора или ячейки памяти, и затем записать команду вызова подпрограммы.

Подпрограммы можно располагать в любом месте памяти, но обычно их помещают сразу после основной программы, что и сделано в нашем примере. В тексте основной программы нашего примера используется команда вызова подпрограммы **CALL ВРЕМЯ**. Эта подпрограмма, как, впрочем, и любая другая, обладает определенной универсальностью. Изменяя значение всего лишь одного операнда, используемого в ней, можно в широких пределах изменять время ее выполнения, а следовательно, на любое заданное время задержать работу основной программы. В нашем примере при первом вызове подпрограммы мы используем операнд **КОНСТ1**, а при втором вызове подпрограммы — **КОНСТ2**. При обращении к подпрограмме значения этих операндов заносятся в регистровую пару **BC** микропроцессора по команде **LXI B, КОНСТ1** или **LXI B, КОНСТ2** основной программы. В определенном смысле подпрограмму **ВРЕМЯ** можно сравнить с одновибратором с перестраиваемой длительностью импульса. В самом деле, мы могли бы подключить светодиод не прямо к выводу порта 1, а через одновибратор, запускаемый положительным фронтом сигнала на выходе младшего разряда порта 1. При этом длительность импульса в 0,25 с на выходе одновибратора задавалась бы *RC*-цепью. Используемый нами «программный одновибратор» по стабильности и диапазону перестройки длительности выходного импульса значительно превосходит своего электронного «двойника».

Теперь рассмотрим, как работает подпрограмма **ВРЕМЯ**. По команде **DCX B** содержимое регистровой пары **BC** микропроцессора уменьшается на 1, а затем в аккумулятор пересылается содержимое регистра **B** и производится операция логического сложения с содержимым регистра **C** этой регистровой пары. Если в регистровой паре **BC** код еще не стал равным 0, то после выполнения этой команды в аккумуляторе окажется число, также отличное от нуля, и выполнится команда условного перехода **JNZ ВРЕМЯ** к началу подпрограммы, все действия повторяются вновь. При этом программисты говорят, что в программе организован цикл. Выход из него возможен только тогда, когда в результате выполнения команды **DCX B** в регистровой паре **BC** окажутся все нули. Тогда работа подпрограммы

закончится выполнением команды **RET** и произойдет возврат к выполнению основной программы. Любая подпрограмма всегда должна оканчиваться командой возврата из подпрограммы.

Вы наверное уже догадались, что временная задержка, обеспечиваемая подпрограммой **ВРЕМЯ**, определяется, во-первых, временем, необходимым для однократного выполнения всех команд этой подпрограммы, и, во-вторых, содержимым регистровой пары **BC**. Последнее и определяет количество программных циклов.

Как же определить число, которое надо поместить в регистровую пару **BC** для задания временной задержки в 0,5 с? Выполнение любой команды микропроцессором занимает строго определенное время. Поэтому, зная длительность выполнения каждой команды, можно вычислить общее время однократного выполнения подпрограммы **ВРЕМЯ**. Оно составляет 9,6 мкс. Следовательно, для задания временной задержки в 0,5 с подпрограмма **ВРЕМЯ** должна быть выполнена $0{,}5/(9{,}6\times\times10^{-6})=52\,080$ раз. Полученный результат необходимо присвоить операнду **КОНСТ2**. Для организации задержки в 0,25 с операнду **КОНСТ1** необходимо присвоить значение, уменьшенное вдвое. При переводе программы в машинные коды и заполнении поля 2 распечатки программы значения второго и третьего байтов команд загрузки регистровой пары **BC** должны быть записаны шестнадцатиричными числами.

В рассмотренном нами примере есть одна часто встречающаяся у начинающих программистов ошибка. Из-за этой ошибки программа может не заработать. В чем же она заключается? Мы «забыли» перед началом работы программы настроить регистр **SP** указателя стека микропроцессора. Настроить — означает поместить в регистровую пару **SP** адрес памяти (ОЗУ), не содержащий кодов команд. Стек используется в нашей программе (команды **CALL** и **RET**), и поэтому мы должны были позаботиться о содержимом регистра указателя стека. Первая команда нашей программы должна была быть командой настройки указателя стека — **LXI SP, СТЕК**. Для стека отведем три ячейки памяти после команды **RET**. Читателю предлагается самостоятельно внести изменения в программу. При этом стоит обратить особое внимание на то, как происходит адресация при работе со стеком (см. «Радио», 1982, № 9).

Теперь мы прервем наше знакомство с программированием до следующего номера, а чтобы время не пропало даром, предлагаем Вам в качестве домашнего задания самостоятельно составить блок-схему алгоритма и написать программу, обнуляющую область памяти, начиная с ячейки с адресом 0100H и до 02FFH. При следующей нашей встрече Вы сможете проверить правильность Ваших рассуждений — мы начнем ее с разбора домашнего задания.

ИНТЕГРАЛЬНЫЕ ОУ В УСИЛИТЕЛЯХ МОЩНОСТИ НЧ

А. СЫРИЦО

ЗВУКОВОСПРОИЗВЕДЕНИЕ

Один из возможных путей существенного упрощения схем усилителей мощности — использование в каскадах усиления напряжения операционных усилителей (ОУ). До последнего времени этому препятствовали два обстоятельства: отсутствие так называемых быстродействующих ОУ, обеспечивающих получение максимальной неискаженной амплитуды сигнала на верхней граничной частоте усилителя НЧ, и недостаточная (из-за ограниченного напряжения источников питания) амплитуда выходного сигнала ОУ в типовом включении.

Первое обстоятельство долгие годы сдерживало применение ОУ не только в усилителях мощности, но и во всех трактах высококачественного воспроизведения и записи звука. Однако в настоящее время нашей промышленностью выпускаются ОУ, быстродействие которых достаточно для использования их в самых различных устройствах предварительного усиления и обработки звуковых сигналов.

Более существенным оказалось второе обстоятельство, и в какой-то мере оно сохраняет свою силу и по сей день. В самом деле, простой расчет показывает, что при использовании ОУ с напряжением питания ±15 В в каскадах предварительного усиления усилителей мощности с типовым выходным каскадом на транзисторах, включенных по схеме с общим коллектором, выходная мощность не будет превышать 6 и 12 Вт при сопротивлении нагрузки соответственно 8 и 4 Ом. Такая мощность явно недостаточна для систем высококачественного звуковоспроизведения.

Известны два способа решения этой проблемы: увеличение (примерно вдвое) амплитуды выходного напряжения ОУ за счет введения следящей обратной связи по цепям питания [1] и применение выходного каскада, усиливающего по напряжению (обычно в 3...4 раза). Однако оба эти способа наряду с достоинствами обладают и существенным недостатком — требуют увеличения числа транзисторов. При использовании второго способа определенные трудности возникают и с защитой транзисторов выходного каскада от короткого замыкания нагрузки.

Анализ свойств ОУ и функциональных возможностей выходного каскада [2] позволил предложить еще один способ получения от каскада предварительного усиления на ОУ увеличенной амплитуды выходного напряжения. Суть способа поясняет рис. 1. Здесь *A1* и *A2* — ОУ, *V3* и *V4* — транзисторы выходного каскада, включенные по схеме с общим коллектором. Питается выходной каскад от двух изолированных друг от друга источников *G2* и *G3*. Увеличение амплитуды выходного сигнала до значения, близкого к удвоенному напряжению источника питания ОУ, достигнуто за счет противофазного возбуждения ОУ *A1* и *A2* (усиливаемый сигнал от источника *G1* поступает на неинвертирующий вход первого из них и инвертирующий вход второго).

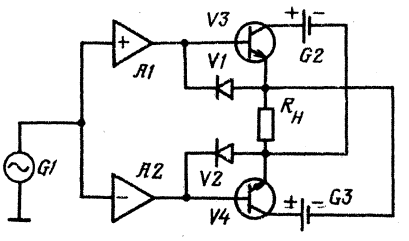


Рис. 1

Предлагаемый вниманию читателей усилитель мощности построен именно на этом принципе. Усилитель обладает рядом особенностей, облегчающих его повторение в любительских условиях. В его выходных каскадах можно использовать любые одинаковые по структуре транзисторы без какого-либо подбора по параметрам. Число переходных конденсаторов сведено в нем к минимуму, причем электролитические конденсаторы исключены полностью. Усилитель допускает возможность изменения в широких пределах сопротивления нагрузки и выходной мощности без дополнительных регулировок. Благодаря применению глубокой ООС по постоянному току отпадает необходимость в установке постоянного нулевого напряжения на выходе усилителя. Усилитель может питаться от нестабилизированного источника. Благодаря хорошей симметрии плеч и отсутствию постоянного напряжения на переходных конденсаторах резкие броски выходного напряжения при включении и выключении питания отсутствуют. Следует