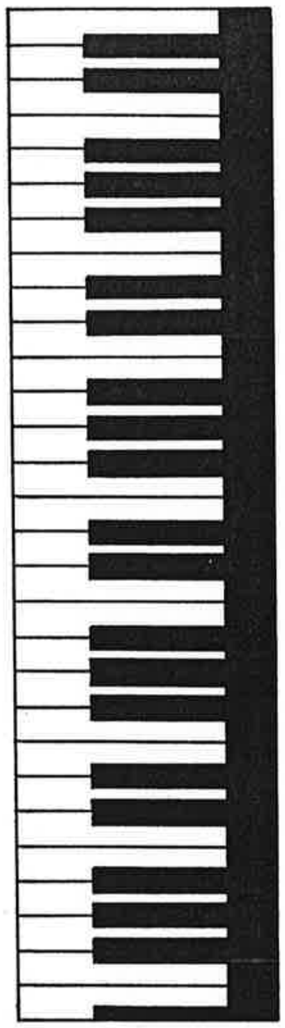
SPECIAL MIDI SUPPLEMENT



THE adoption of MIDI by all the major electronic musical instrument manufacturers, and the inclusion of MIDI sockets on an increasingly wide range of equipment, has certainly revolutionised electronic music making. MIDI stands for "Musical Instruments Digital Interface", and it is a means of exchanging information between electronic musical instruments, or any equipment related to electronic music production.

Although MIDI is regarded by many as nothing more than a modern alternative to the old gate/CV method of interfacing synthesisers, or a means of synchronising drum machines, it actually has capabilities which go well beyond this. This supplement is designed to help you exploit some of the many MIDI facilities, or to simply start to use the MIDI facilities in conjunction with a BBC Micro.

The first article describes a **BBC/MIDI Interface** by **Mike Hughes.**

QUITE a number of medium priced electronic musical keyboards are now provided with MIDI (Musical Instruments Digital Interface) circuitry but, unless you are into music in a big way or are prepared to spend a lot of money on special hardware interfaces, it is possible that you have never experimented with that very sophisticated piece of hardware lying dormant in your instrument (If you already have experience of MIDI the following projects—MIDI MERGE and MIDI PEDAL—will no doubt interest you).

If you have a BBC Model B computer you can, for the outlay of a few pence and a bit of software, turn it into a controller for your keyboard. With a suitable supporting program (which can be written in BASIC or better still in Assembler) you can get your computer to play polyphonic tunes, sequence rhythm patterns or provide a vamping bass whilst you practice with the melody.

This article describes the hardware interface required and gives a short program listing to provide the fundamentals on to which you can build your own more specialised software

MIDI MIGHT BE THERE

You can tell if your keyboard is equipped for MIDI interfacing by looking for a couple of innocent five pin DIN sockets labelled MIDI IN and MIDI OUT. Because of the simplicity of our interface you will only be able to drive the keyboard from the computer so we shall be mainly interested with the MIDI IN socket.

If your keyboard is equipped with MIDI you may not be aware that whenever you press a key (or a set of keys) the instrument sends out three serial 10 bit digital codes to the MIDI OUT socket describing each key that is pressed or released. These codes would be used to control another keyboard or a synthesiser etc.

Likewise the keybord is constantly checking the MIDI IN socket to see if similar sets of codes are being transmitted to it from another instrument or, as we shall be doing, from a computer.

The code numbers describe such things as:

a) Is a key being pressed and, if so, which one.
b) Has a key just been released and, if so, which one.
c) Which instrument should respond to the signal (if several instruments are connected together).
d) For touch sensitive keyboards—how hard the key was hit.

This list is by no means fully comprehensive of the whole set of MIDI instructions but these are the relevant ones as far as this article is concerned.

HIGH SPEED IS VITAL

If you imagine that for every key pressed or released three lots of ten bits of data (30 bits in all) have to be transmitted then you will appreciate that the data has to be transmitted at a very high speed to cope with fast moving musical chords. The alternative would be for parts of the music to be left out which would, of course, be totally unacceptable.

To cater with this situation MIDI serial data has to be transmitted at the unusually high rate of 31.25 kilobaud which means 31,250 bits per second. This rate was chosen as a reasonable compromise between the fastest speed of playing, practical communications technology, and also because the 31.25 kilobaud clock rate can be obtained by a simple binary division from 1, 2 or 4 MHz. For example 2 MHz divided by 64 provides a frequency of 31.25 kHz.

This baud rate ensures that a 10 bit serial word will be transmitted in 320 microseconds and the group of three codes (to define a "key down" or a "key up" movement) in less than a thousandth of a second—fast enough for most styles of playing even if chords are involved!

THE SERIAL FORMAT

The basic equipment of MIDI transmission is the 10 bit word made up of a start bit at logic level "0" followed by 8 bits of data (which is the code number in question) and terminated with a single stop bit at logic level "1". The start bit and bits of the data each take 32 microseconds to transmit and the connecting wire goes to logic level "1" for the stop bit and stays at that level until the beginning of the next word.

Conceptually it is a simple matter to make a computer output a sequence of "ones" and "zeros" at an output port and provided that the computer is fast enough and the program runs at a controlled speed it is fairly easy to generate this basic MIDI serial word.

THE BBC COMPUTER IS FAST ENOUGH

Fortunately the BBC Model B has a fast enough clock to provide the 31.25 kilobaud rate required (provided certain precautions are taken) but the program has to run extremely fast. For this reason the transmission software has to be assembled into machine code. Unfortunately, for this application, the BBC computer makes extensive use of interrupts which break into any other operation taking place. If this was allowed to happen during the transmission of a word the baud rate timing would be disrupted. It is, therefore, necessary to disable the computer's interrupts whenever transmission takes place.

The BBC machine has a convenient User Port which can be used to output the data but the signal levels at this port are normal TTL voltages and this does not exactly meet the input characteristics of a MIDI IN socket. To appreciate the difference it is useful to understand the standard MIDI input hardware specification which is based on a 5mA "Current Loop".

CURRENT LOOP AND OPTO ISOLATION

To minimise earth loop problems (which give rise to mains hum in complex interconnected audio systems)—and also to avoid expensive damage by wrong connections—the keyboard's MIDI IN socket is protected by an optical isolator consisting of an l.e.d. optically coupled to a photodiode. The transmitted data activates the l.e.d. inside the keyboard and the photodiode detects the changes and passes the digital signals to the rest of the keyboard's circuitry. The l.e.d. requires a current of 5mA to illuminate and this current is derived from the source of transmission which, after flowing through the l.e.d., returns to the transmission source—hence the term "Current Loop".

The input specification requires the l.e.d. to be illuminated for a transmitted logic level "0". This means that during a stop bit (and whilst waiting for the next word) the l.e.d. is extinguished.

CIRCUIT

The BBC's User Port is not directly capable of providing this 5mA current drive. It is, however, a very simple matter to convert the TTL voltages into a current of

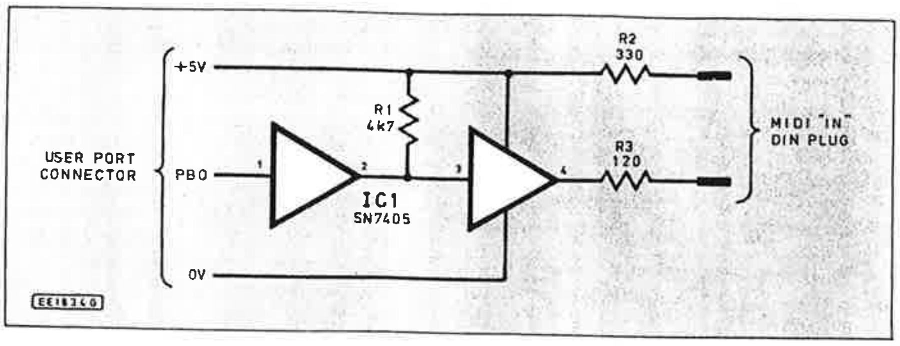


Fig. 1. Circuit of the BBC/MIDI Interface.

5mA which has the right polarity and logic sense to match the MIDI input requirement. This converter is the only piece of hardware required and it can be made out of two inverters from an SN7405 (standard TTL hex. open collector inverter) and three resistors as shown in the schematic drawing (Fig. 1). The prototype was made up on a small piece of matrix board glued to the back of a 20 pin User Port plug. Power is taken from the User Port socket itself.

The converter's output is fed down a screened lead (not to exceed 50 feet in length) to a standard 5 pin (180 degree) DIN plug. It is vitally important that the +5V rail, through R2, goes to pin 4 of the DIN plug. This is the pin identified by the figure 4 moulded into the base of the DIN plug. The inverter's output, through R3, goes to pin 5 of the DIN plug. If you get these connections the wrong way round current will try to flow the wrong way through the keyboard's l.e.d. and no signals will be received!

MACHINE CODE TRANSMISSION ROUTINES

The Assembler Language listing contains four machine code sub-routines which are designed to reside in memory well out of harms way below PAGE. These, once loaded, can be called from any BASIC program as required.

When SETPORT is called (use the statements CALL SETPORT or CALL &C00 depending on whether SETPORT has been defined as a variable) it simply defines the least significant bit of the User Port (PB0) to be an "Output Bit". This should

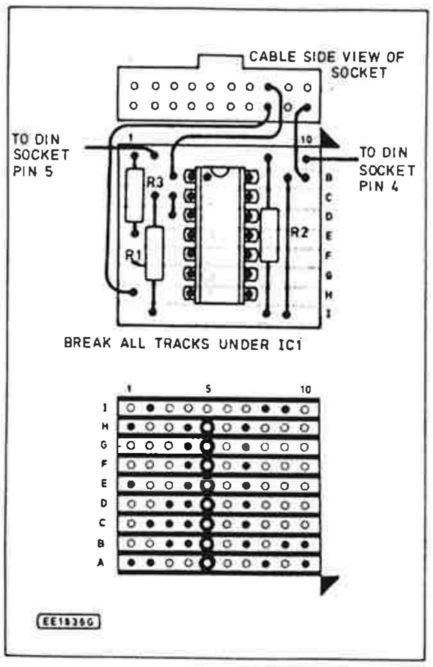


Fig. 2. Construction of the MIDI Interface

be one of the first things you do in your control program.

SEND is the main transmission sub routine and before calling it from BASIC you must first of all set the variable A% to the code number that is to be transmitted. When SEND is called the contents of A% are automatically transferred to the microprocessor's Accumulator. The statement to use is CALL SEND or CALL &C0B.

The first thing SEND does is temporarily store the contents of the accumulator on to the stack and sets the User Port output to zero (Lines 170-190). This creates the beginning of the serial start bit which has to be held for 32 microseconds. This precise time delay is produced by the call to DELAY in Line 200 followed by a NOP which is a small timing correction factor.

Lines 220 and 230 recover the original data off the stack and get ready to transmit the eight bits in serial fashion. Sequential transmission of the eight bits is done by Lines 270-310 by rotating the accumulator right and sending the result to the User Port eight times. Each time this is done a delay of 32 microseconds is generated. In this fashion each bit from the accumulator (starting at the least significant bit) is output from User Port PB0 and, from the outside world this looks like a serial stream of data.

FINISH completes the sub routine by outputting logic level "1" (the stop bit) at

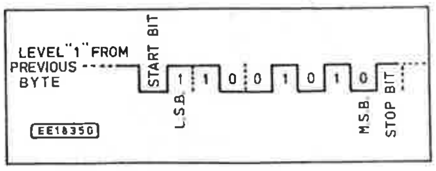


Fig. 3. Format of a MIDI serial byte of data.

the User Port. This is, once more, held for at least 32 microseconds before the sub routine returns to BASIC.

The remaining two sub routines are IOFF and ION which respectively switch off the internal interrupts of the BBC computer (to ensure precise timing of the SEND routine) and restore them. In practice these could be incorporated directly into the beginning and end of the SEND routine but have been left as separate callable routines for versatility. It is essential that IOFF is called by BASIC (use CALL IOFF OR CALL &C31) before SEND is called and, for convenience, ION should be called immediately after the return from SEND (use CALL ION or CALL &C33).

ODES YOU HAVE TO SEND

There are a large number of MIDI control codes (all of which have very useful purposes) but the sheer quantity of them, and complexity of some of them, can be a bit awe inspiring. Furthermore some codes are not supported by some keyboards and this can often give rise to confusion. For this reason we shall limit ourselves to considering just two codes which, by their very definition, have to be universal to all MIDI keyboards. These are the codes which tell a note to start sounding and stop sounding—how fundamental can you get!

Do not scoff at such a simple approach because you can do an awful lot and produce spectacular music with just these two code sequences.

Each of these two codes is made up of three separate 8 bit bytes of data. The first byte defines the operation together with the channel numbers (i.e. the identification number, or address, of the instrument which is to receive the data); the second byte defines the number of the note (Middle "C" is always note number 60 in MIDI parlance and other note numbers can be worked out by adding or subtracting 1 for each semi-tone up or down from Middle "C" so Top "C" is note 72 and Bottom "C" in note 48. Likewise Middle "G" is note 67 and so on).

The third byte defines the rate at which the key was hit or released—this is specifically for touch sensitive keyboards but we shall use the default values which will apply to non touch sensitive instruments.

THE "NOTE ON" SEQUENCE

To make a specific note start sounding you have to send the three byte code described above. The first byte is made up of the following binary pattern:

1001xxxx

where **xxxx**, the least significant "nibble", defines the Channel Number that the receiving instrument is set to. Here is the first bit of potential confusion; the binary nibble 0000 is zero but it is used to define Channel 1 on the receiving instrument and although 1111 in binary means 15 in denary it defines Channel 16. Fortunately we don't have to worry about this because, to make our system fairly foolproof, we shall operate in MIDI's "Omni" mode which means that ANY instrument will respond to Channel 1 unless it has been specifically set to something different.

In more simple terms the first byte is set to the decimal number 144 plus the binary version of the Channel Number. In the case of Channel 1 we simply have to add zero.

The second byte is the number of the note—as already described. It can be any value between 1 and 127 with 60 describing Middle "C". In practice five octave organ keyboards range from about 36 to 96 whereas full size piano keyboards range from about 21 to 108.

The third byte describes the rate of pressing a key but for non touch sensitive keyboards you should use the value of 64.

Therefore, to make Middle "C" start sounding you have to send the following decimal numbers to the keyboard:

144,60,64

THE "NOTE OFF" SEQUENCE

The sequence which stops a note sounding follows the same sort of pattern that is used to start a note except that the first byte is constructed as follows:

1000xxxx

i.e. 128 plus the binary equivalent of the Channel Number.

The second byte defines the note which is to stop—exactly as before—and the third byte is the key release velocity which, for non touch sensitive keyboards, should always be zero.

The code sequence to stop Middle "C" sounding should, therefore, be:

128,60,0

PLAYING A CHORD

To play a chord of "C Major" requires the Middle notes C, E, G and Top C to be played simultaneously and held down for the duration of the chord. In long hand the MIDI sequence to be transmitted in words and numbers would be:

Start Middle C	144,60,64
Start Middle E	144,64,64
Start Middle G	144,67,64
Start Top C	144,72,64
Wait for the duration of the chord	
Stop Middle C	128,60,0
Stop Middle E	128,64,0
Stop Middle G	128,67,0
Stop Top C	128,72,0

Note that it is very important to keep a record of notes which have started so that you can stop them at the required point in time. They will not stop unless you tell them to and that can lead to terrible dischords!

HOW TO USE THE DRIVER ROUTINE

The simplest way to get used to MIDI interfacing and developing programs is to start in an elementary way. First of all type in the Assembly Language program for the driver and save it to disc or tape before running it. Run it and the machine code will be set up in memory from location &C00. Clear the memory by typing NEW and then, after connecting the current loop driver between the computer and your keyboard, try making Middle "C" sound for a short while by entering and running the following short BASIC program (**NOTE** Don't type in the comments):

```
10 CALL &C00      ' Setup the output port
20 CALL &C31       Switch off interupts
30 A%=144          Set start note byte & Channel
40 CALL &C0B       Transmit it
50 A%=60           Define Middle C
60 CALL &C0B       Transmit it
70 A%=64           Define key on velocity
80 CALL &C0B       Transmit it
90 FOR X=1 TO 500: NEXT X  Note duration
100 A%=128         Set note off byte & Channel
110 CALL &C0B      Transmit it
120 A%=60          Define Middle C
130 CALL &C0B      Transmit it
140 A%=0           Define key off velocity
150 CALL &C0B      Transmit it
160 CALL &C33      Enable interupts
```

Take particular note of Line 10 which sets up the configuration of the output port and Lines 20 and 160 which, respectively, disable and enable the computer's interrupts.

There are obviously shorter cut methods of writing such a fundamental program which has such a large number of repeated operations but we have written it in sequential form to emphasise the order in which the codes have to be transmitted.

A shorter, and more versatile program could be written around DATA statements using an unused code, like 255, to define the pause. For example the following program plays the chord of C Major:

```
10 CALL &C00
20 CALL &C31
30 FOR X=1 TO 25
40 READ A%
50 IF A%=255 THEN FOR Y=1 TO 500: NEXT Y:
GOTO 70
60 CALL &C0B
70 NEXT X
80 CALL &C33
90 END
100 DATA 144,60,64,144,64,64,144,67,64
110 DATA 144,72,64,255,128,60,0,128,64,0
120 DATA 128,67,0,128,72,0
```

We chose a program to play a static chord to show that the system is polyphonic and there is, in theory, no limit to the number of notes which can be played simultaneously.

CHROMATIC SCALE

The following example shows how you can play a single note chromatic scale between Middle C and an octave above Top C:

```
 10 CALL &C00: REM Set up output port
 20 CALL &C31: REM Disable interupts
 30 FOR X%=60 TO 84
 40 PROCtransmit
 50 NEXT X%
 60 CALL &C33: REM Enable interupts
 70 END
 80 DEFPROCtransmit
 90 A%=144: CALL &C0B
100 A%=X% : CALL &C0B
110 A%=64 : CALL &C0B
120 FOR Z=1 TO 100: NEXT Z
130 A%=128: Call &C0B
140 A%=X% : CALL &C0B
150 A%=0  : CALL &C0B
160 ENDPROC
```

TAKING IT FURTHER

Use of the other MIDI codes is beyond the scope of this particular article but we hope we have wetted your appetite. Clearly a lot of enjoyment can be gained from controlling your keyboard from software and the largest part of such a project is developing software that can make the most of the interface.

For those who would like to go further (but do not have programming skills) a complete compiler program is available on 40 track DFS disc or tape which allows you, in a very simple and speedy way, to write your own polyphonic music or backing, save it back to disc or tape and play it through your keyboard.

The software has been specially written to support this article but is too lengthy for printed publication. Your tune, which can have 12 note polyphony, is speedily written directly on to the screen and is held as machine code which allows long compositions and also permits repetitive sections to be looped or played at varying tempos. The program also permits keyboard voices to be changed while the music is playing. Included with the programme are a couple of sample tunes.

Copies of the program and instructions can be obtained direct from Mike Hughes at 2 Oaklands Lane, Biggin Hill, Westerham, Kent, TN16 3DN. Please state clearly whether you want Disc or Tape when you order with your remittance of £10 for disc or £7 for tape (including post and packing).

LISTING OF FUNDAMENTAL DRIVER ROUTINES

The following program produces the machine code sub routines which, as described in the main article, output a MIDI 10 bit word (one start bit, eight data bits and one stop bit).

The program should be typed in and saved to disc or tape before running it. When you run it the code is assembled from location &C00.

You can, if you wish, use this listing to precede any BASIC program which you generate yourself. In this case your basic program should begin at Line 630 and you will have the advantage that you will not need to call the sub routines at their numeric addresses; you would simply have to use CALL SETPORT, CALL SEND, CALL IOFF or CALL ION as all the address variables will have previously been defined by the previous part of the program.

Remember that this program is only of use if you use the current loop interface between your BBC Model B and your keyboard.

```
 10 FOR opt%=0 TO 3 STEP 3
 20 P%=&C00
 30  [
 40           OPT  opt%
 50 :
 60 \ Call SETPORT to define port config
 70 :
 80 .SETPORT LDA  #1
 90           STA  &FE62
100           LDA  #1
110           STA  &FE60
120           RTS
130 :
140 \ Put transmit data in A% and Call SEND.
150 \ SEND transmits single start bit
160 :
170 .SEND     PHA
180           LDA  #0
190           STA  &FE60
200           JSR  DELAY
210           NOP
220           LDX  #8
230           PLA
240 :
250 \ BYTE runs on to transmit data in A%
260 :
270 .BYTE     STA  &FE60
280           JSR  DELAY
290           ROR  A
300           DEX
310           BNE  BYTE
320 :
330 \ FINISH transmits single stop bit
340 \ and returns you to BASIC
350 :
360 .FINISH  LDA  #1
370           STA  &FE60
380           JSR  DELAY
390           RTS
400 :
410 \ DELAY loop of 8 gives precise baud rate
420 :
430 .DELAY    LDY #8
440 .LOOP     DEY
450           BNE  LOOP
460           RTS
470 :
480 \ Call IOFF before calling SEND
490 \ Interupts have to be disabled to
500 \ ensure correct baud rate
510 :
520 .IOFF     SEI
530           RTS
540 :
550 \ You must call ION to restore
560 \ interupts after return from SEND
570 \ otherwise keyboard will be locked out
580 :
590 .ION      CLI
600           RTS
610 ]
620 NEXT
```

COMPONENTS

Shop Talk
See page 193

Resistors

R1	4k7
R2	330
R3	120

All ¼W carbon

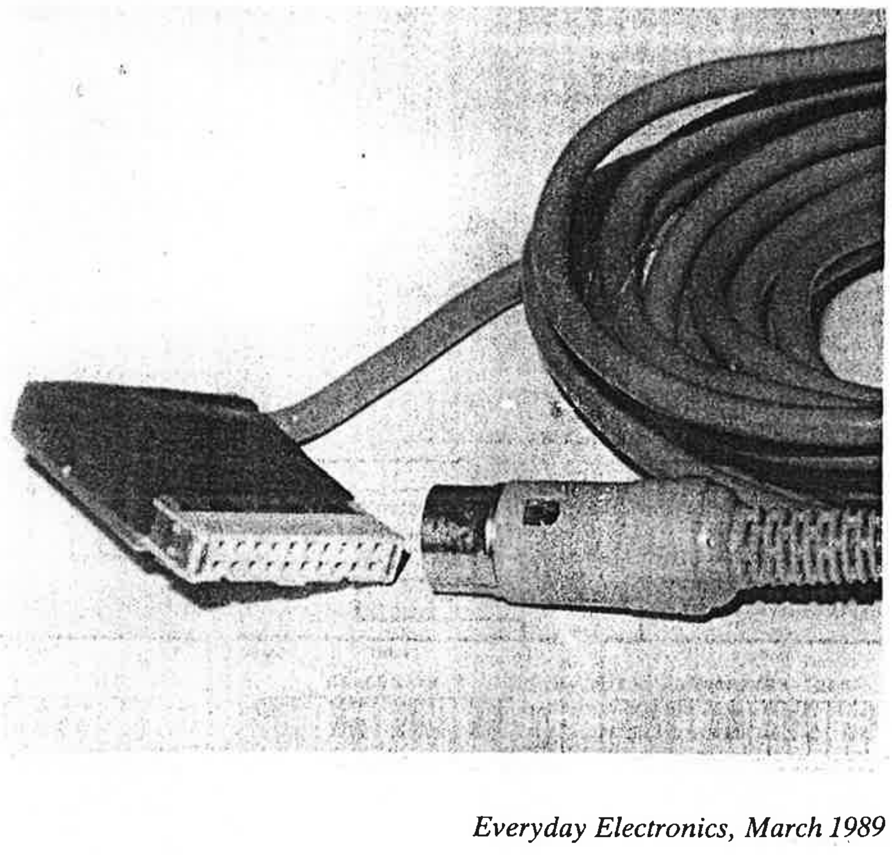
Semiconductors

IC1	SN7405 Hex inverter

Miscellaneous

Plug to suit the BBC Micro User Port (20 pin); five way 180° DIN plug; Veroboard, 9 strips by 10 holes, 0.1 inch matrix; connecting cable.

Approx. cost Guidance only **£4** *plus software*



SIMPLE MIDI MERGE UNIT

Robert Penfold

IN THE pre-MIDI era sophisticated electronic music systems were something of a rarity. These days sophisticated computer controllers and polyphonic multi-timbral instruments have much more modest price-tags, and are generally vastly superior to the top-flight instruments of a few years ago. With a few pieces of MIDI equipment there are almost limitless musical possibilities.

There are also a large number of ways in which a given set of MIDI units can be organised into a system. In some cases what might be quite desirable might also be impossible due to a lack of MIDI ports to permit everything to be fitted together in the desired fashion. Usually a very simple add-on is all that is needed in order to facilitate the required method of connection. The two standard add-ons for these situations are MIDI THRU boxes and merge units. A THRU box (as described in the July 1987 issue of *Everyday Electronics*) simply takes a MIDI input signal and splits it to provide two or more buffered outputs. A MIDI merge unit provides the opposite function, and combines two MIDI signals into one.

A sophisticated MIDI merge unit will operate with signals on both inputs simultaneously. It takes signals on one input and passes them through to the output, while signals on the other input are stored in a buffer until the output is free and they can be transmitted. Provided the two sources do not provide so much data that the unit becomes overloaded, this gives good results with no significant time-shifting of any MIDI messages.

While advantageous in some circumstances, a merge unit of this type is quite complex and expensive as it needs to be microprocessor based. For most purposes something that simply couples two inputs through to a single output will suffice, but simultaneous operation on both inputs must be avoided. This would give a scrambled output that would result in a malfunction of the system. A unit of this type should really be regarded as an alternative to a MIDI switch, but it has the advantage of effectively being automatic in operation.

USES

A simple application for a MIDI Merge Unit would be in a setup of the type depicted in Fig. 1. Here a synthesiser (which could be a rack mounted type rather than a keyboard instrument) is played from a MIDI keyboard. The latter could be a synthesiser with the second instrument acting as a "slave" to it, or it could be just a MIDI keyboard.

Suppose that the synthesiser will sometimes be used with a step-time sequencer such as a computer running a notation program. Either a lot of plugging and unplugging will be needed, or some form of switch or merge unit is required to feed signals from the desired controller to the MIDI "IN" socket of the synthesiser. As explained previously, a merge unit has the advantage of effectively giving automatic switching. It can be connected up and then largely forgotten!

Although it might seem to be possible to obtain much the same effect by feeding the output of the sequencer to the input of the keyboard, this will not work. The input signal appears on the "THRU" output and not at the "OUT" socket (apart from a few instruments which can be switched to this non-standard mode of operation). Driving the synthesiser from the "THRU" output of the keyboard will not work either. The keyboard's output signal does not appear at this socket.

Another arrangement is shown in Fig. 2 in which a simple merge unit can be used to good effect. Here two synthesisers have been "chained" together and are being played from a separate keyboard. A MIDI pedal unit is used to provide program change messages to switch to different sounds from the synthesisers at the appropriate times. A merge unit enables both the keyboard and the MIDI pedal to drive the synthesisers, but with a simple merge unit of the type described here it is necessary to take care not to operate the pedal and the keyboard simultaneously. Once again, with most keyboards it is not possible to obtain the desired effect by feeding the output of the MIDI pedal to the keyboard's "IN" socket.

As a point of interest, most MIDI keyboards have a "local off" mode, where the keyboard is disconnected from the sound generator circuits. The keyboard still functions and drives the MIDI "OUT"

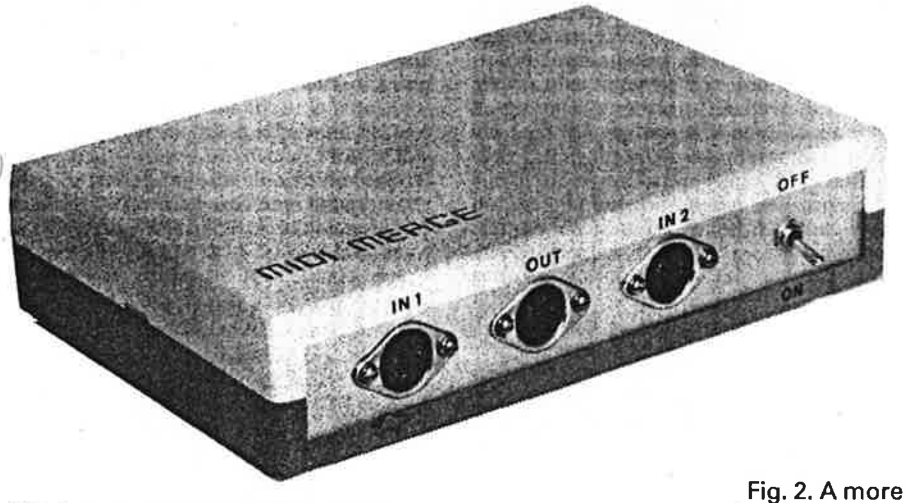


Fig. 1. A simple MIDI system using a merge unit.

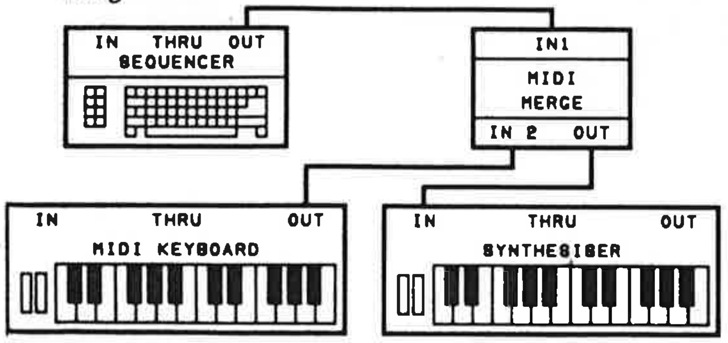
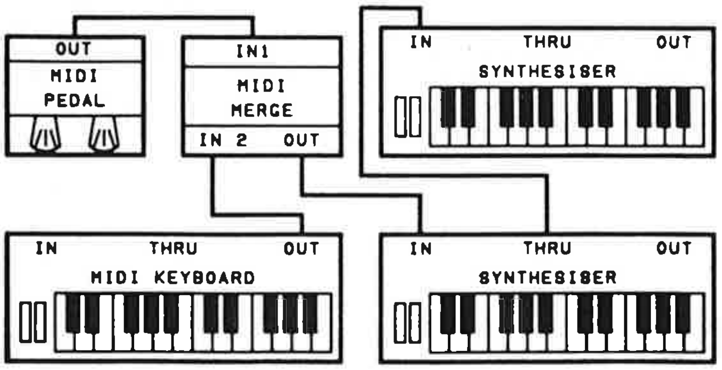


Fig. 2. A more complex setup using a merge unit.



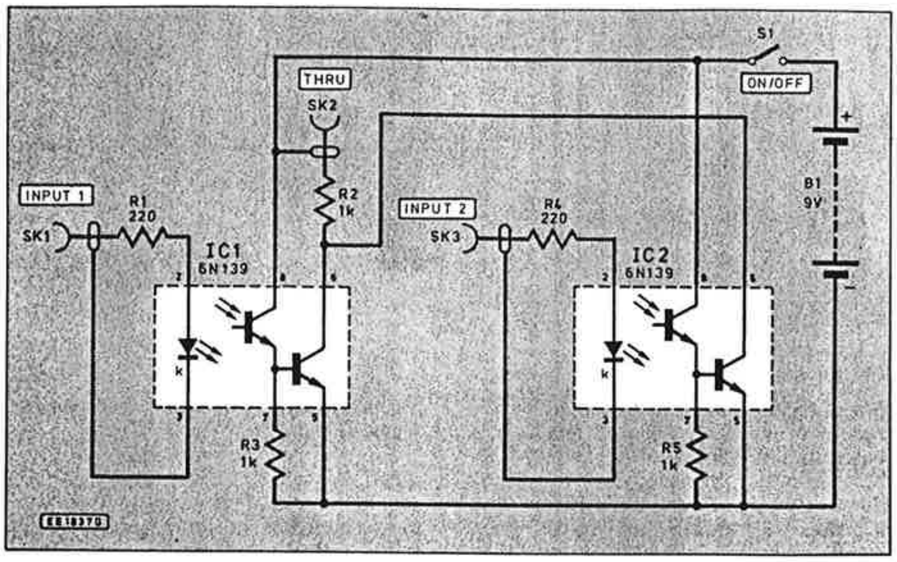


Fig. 3. The simple MIDI Merge Unit circuit diagram.

socket, and the sound generator circuits still respond to information received via MIDI. The instrument is effectively turned into a separate keyboard and MIDI sound module. The setup of Fig. 2 could therefore be implemented using a synthesiser switched to the "local off" mode to act as the separate keyboard and one of the synthesisers.

THE CIRCUIT

A unit of this type can be very simple indeed, as will be apparent from the circuit diagram of Fig. 3. MIDI outputs are generally open collector types, and a simple merge function could conceivably be obtained simply by connecting two outputs in parallel. As the exact forms of output stages used are unknown quantities, and it is quite likely that none of the MIDI ouput terminals will be at earth potential, this would be more than a little risky, and is certainly not to be recommended. It is better to have a simple combiner circuit, such as the one described here, which has an opto-isolator at each input. This ensures complete isolation between the two sources.

The MIDI baud rate of 31.25 kilobaud is high enough to make it difficult to feed the signal through a relatively slow component like an opto-isolator and obtain a reasonably accurate reconstruction of the signal at the output. Ordinary opto-isolators of the type which use an infra-red l.e.d. and a photo-transistor seem to be totally inadequate, and even the high efficiency/high speed types seem to be barely adequate for this application.

In this circuit a slightly different type of opto-isolator is used, and this is a component which has one transistor in the emitter follower mode, driving another which acts as a common emitter switch. This configuration might look rather like the familiar Darlington Pair arrangement, but it is not, as the collector of the two transistors are not connected together. Darlington devices offer good sensitivity but are very slow, and totally unsuitable for this application.

In fact the configuration used here is too slow without the inclusion of the emitter load resistor for the first transistor in each opto-isolator (R3 and R5). This ensures that the emitter follower stage operates at a reasonably high current, and that it operates suitably fast. Using a 16kHz squarewave test signal (which is comparable to a MIDI signal) there is no significant degradation of the waveform through the unit. In theory this circuit should operate at something like ten times the MIDI baud rate, and it should give excellent reliability with MIDI signals.

OUTPUT

On the output side of the circuit, the two open collector outputs are simply wired together, and connected to the output socket via a common current limiting resistor (R2). Note that current limiting resistors are included at each input, as called for by the MIDI standard. This is presumably to protect the l.e.d.s in the event of equipment being connected incorrectly.

Power is obtained from a nine volt battery. This should have a long operating life as the maximum current consumption (with a continuous stream of data) is only about 2.5 milliamps, and the current consumption under stand-by conditions is likely to be no more than a few microamps. If the unit should happen to be accidentally left switched on for long periods it is unlikely that the battery will discharge to any significant extent.

CONSTRUCTION

Details of the printed circuit board are shown in Fig. 4. Construction of the board is very straightforward, but do not overlook the single link-wire. The opto-isolators are not static sensitive devices, but as they are not particularly cheap I would recommend that they are fitted into sockets. At this stage only fit pins to the board at the points where the connections to off-board components will eventually be made.

I used a case having approximate dimensions of 180 by 120 by 40 millimetres, but the unit could be fitted into a much smaller case if desired. The three sockets and the on/off switch are mounted in a row along the front panel of the unit. The input and output sockets are five way (180 degree) DIN types, which are the standard MIDI connectors. Each socket requires two short round-head 6BA fixing screws plus matching nuts. The printed circuit board is mounted on the base panel of the case using M3 or 6BA mounting screws plus spacers, or plastic stand-offs.

COMPONENTS

Shop Talk
See page 193

Resistors

R1, R4	220 (2 off)	
R2, R3, R5	1k (3 off)	

All 0.25 watt 5% carbon.

Semiconductors

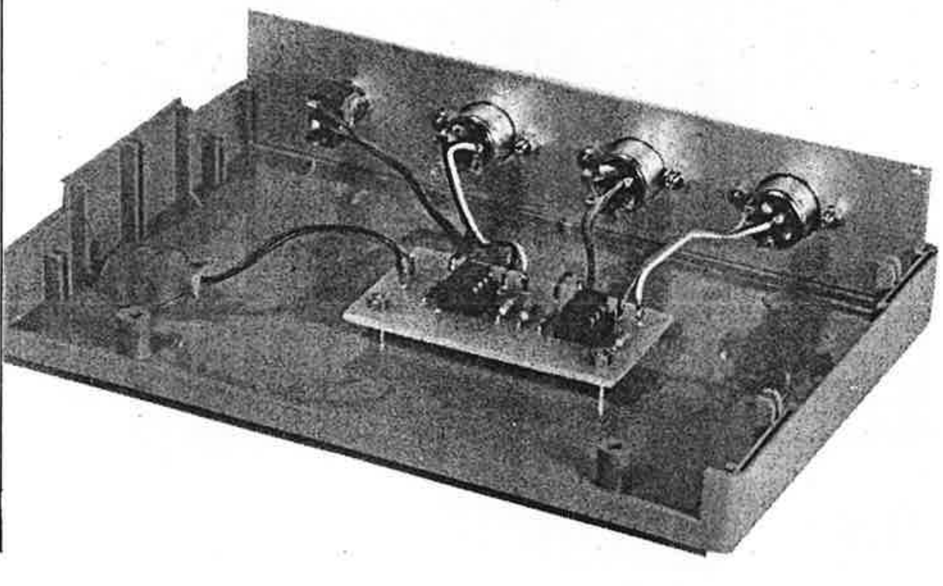
IC1, IC2	6N139 Darlington opto-isolator (2 off)

Miscellaneous

SK1 to SK3	5 way 180 degree DIN sockets (3 off)
S1	S.P.S.T. sub-miniature toggle switch
B1	9 volt battery (PP3 size)

Printed circuit board available from *EE PCB Service* order code EE640; case about 180mm × 120mm × 40mm; battery connector; pins; wire; solder; etc.

Approx. cost Guidance only **£14**



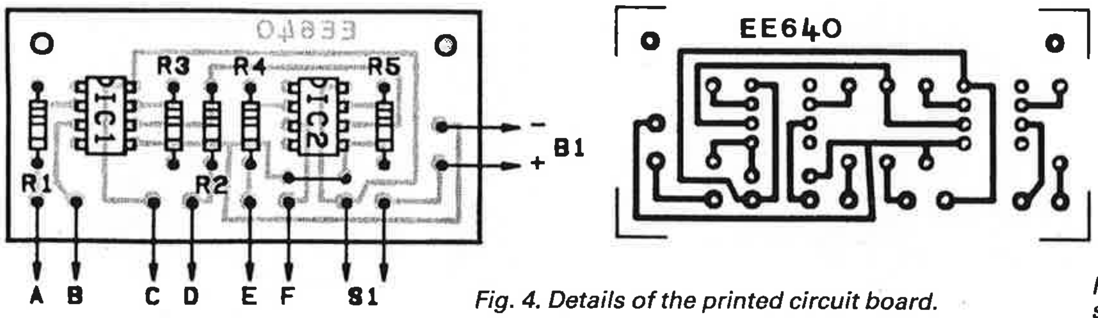


Fig. 4. Details of the printed circuit board.

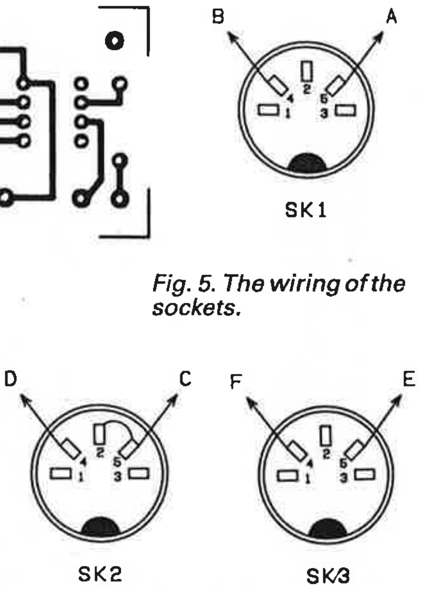


Fig. 5. The wiring of the sockets.

The small amount of hard-wiring is then added, using ordinary insulated multi-strand connection wire. Fig. 5 shows the wiring to the three sockets. This method of connection enables the unit to be wired into a system using standard MIDI cables. If you wish to make your own cables, twin screened lead is required.

The inner conductors are used to connect pins 4 and 5 on one plug to the same pins on the other plug. Cross-over connection (i.e. connect pin 4 on one plug to pin 5 on the other) must not be used. Apparently some ready-made audio DIN leads do use cross-over connection, and are consequently unsuitable for MIDI use. The cable's screen is used to connect pin 2 on one plug to pin 2 on the other plug.

TESTING

In order to test the unit it is merely necessary to connect a MIDI controller of some kind to each input, and to couple the output to a MIDI instrument. With the merge unit switched on, operating either controller should then produce the appropriate result from the instrument. If not, switch off the merge unit and recheck all the wiring. Also, make sure that the MIDI equipment is all operating on suitable modes, channels, etc. □

MIDI PEDAL

Robert Penfold

THIS MIDI Pedal project exploits one of the extra MIDI facilities; the program change messages. These messages are one of the more versatile aspects of the MIDI system, and this unit can be used to good effect with a wide range of MIDI equipment.

QUICK CHANGE

There are two basic types of MIDI message, which are the channel and system types. The note on/off messages are the most common channel types, and these carry a channel identification number so that they can be directed to one particular instrument, or even to one voice of one particular instrument in the system. The system messages, which include the so-called MIDI clock timing signals, are directed to everything in the system, and cannot be targeted at one particular item of equipment or voice of an item of equipment.

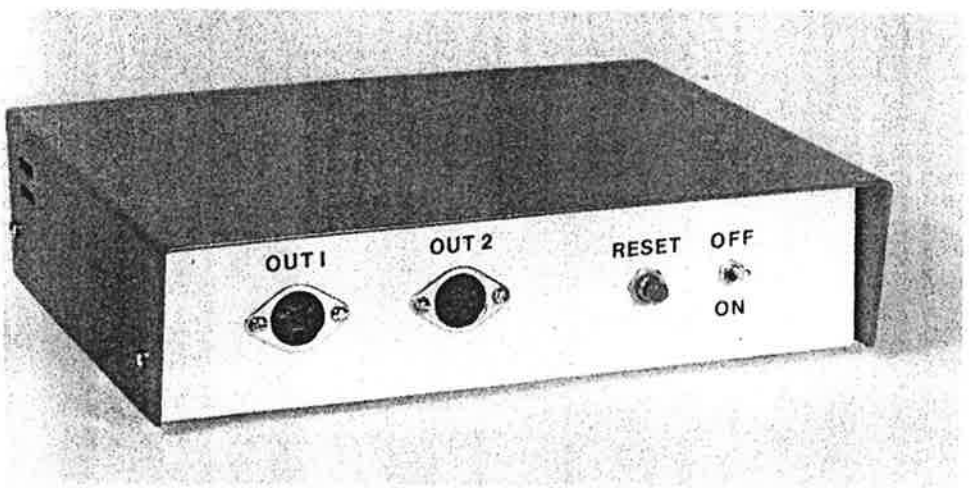
One of the most useful of the MIDI channel messages is the program change type. MIDI messages always start with a header byte which contains the code for the message involved, plus the channel number where appropriate. This header byte is then followed, where necessary, by one or more data bytes. The program change message has just one data byte, which is the new program number in the range 0 to 127.

Note that MIDI distinguishes between instruction and data bytes by having the most significant bit of instruction bytes always set to 1, and those of data bytes always set to 0. This leaves only seven bits left for data, and gives a range of 0 to 127 rather than 0 to 255.

Some parts of the MIDI specification are rigidly defined, while other parts are left extremely vague. While I suppose that the vagueness can be a drawback, it does permit great versatility. The program change message is a good example of this. Each program is really a collection of control settings rather than a program in the normal computer sense. The MIDI specification does not lay down any rules as to what controls can be included under the command of program change messages, and this is entirely at the discretion of the equipment designers.

This has the drawback that just what each program number represents varies from one piece of equipment to another. It could even be different for two items of gear of the same type, because one might have been set up differently to the other. It gives great versatility though, since virtually anything can be controlled via MIDI program change messages.

In most cases these messages are used with instruments where each program is a different group of settings for the envelope shapers, filters, etc. In other words, each program produces a different sound from the instrument, and changing from program 1 to program 2 could switch from a piano to an organ sound for instance. It is important to realise that each program is a series of preset control settings, and that



program change messages do not permit adjustment of individual controls (which is possible via the control change messages). However, for many purposes switching from one range of settings to another is all that is required.

MIDI UNITS

There are now a wide variety of MIDI controlled units available, including effects units and audio mixers. These are mostly controlled via MIDI program change messages. Taking an effects unit as an example, this could be fed from the MIDI OUT socket of a synthesiser, and the latter could be set up to transmit program change messages each time its program was changed by way of the front panel controls. The effects unit would then change programs in sympathy with the synthesiser, and it could be set up to provide a suitable effect to enhance each sound produced by the synthesiser.

A MIDI pedal generates a program change message each time the foot-switch is operated. The most basic way of using a MIDI pedal (but a very effective one) is to have it driving the MIDI IN socket of a synthesiser or other MIDI equipped instrument. This permits the "no hands" approach to changing sounds. There are numerous other ways in which a MIDI pedal can be utilized though. It could be used to drive an effects unit, or both an instrument or an effects unit, or any equipment that responds to program change messages for that matter.

This MIDI pedal simply increments the program number by one each time the unit is activated. The program would presumably be set at 0 initially, but whatever the initial setting, the first operation of the pedal switches it to program 1, the next switches it to program 2, and so on. This method enables the electronics of the pedal unit to be kept reasonably uninvolved, but it is compatible with most MIDI equipment where any desired control settings can usually be assigned to each program. It is not of much use with any items of equipment that have factory preset programs which are not user redefinable.

In order to obtain worthwhile results it must be possible to set up the programs of the instrument so that the required sounds are obtained as the pedal is used to take the instrument through a sequence of programs. However, I have not encountered any units of the purely preset type, and most MIDI equipment seems to be designed to make it reasonably easy to set up each program in the desired manner.

In many cases the pedal will be used with equipment set to the "omni" mode, where messages on all the MIDI channels are accepted and responded to. The pedal unit can be preset via an "on-board" hex switch to transmit on any one of the sixteen MIDI channels, and is therefore usable in a setup where operation on a specific channel is essential. The unit has two MIDI OUT sockets, and it is self contained with power being provided by an internal nine volt battery.

SYSTEM OPERATION

MIDI is a serial system which is very much like the standard RS232C type. It differs from the RS232C system in that it operates at the relatively high baud rate of 31250 baud, which is not a standard RS232C rate. Also, it does not use ordinary RS232C signal levels. In fact it is a current loop system which has an opto-isolator at each input. It operates at a nominal current of five milliamps.

The arrangement used in this unit is shown in Fig. 1, albeit in somewhat simplified form. The UART (universal asynchronous transmitter/receiver) is at the heart of the unit, and it is this that generates the basic serial output signal. UARTs are better suited to this type of unit than most other serial devices, as other types are almost invariably only suitable for microprocessor based circuits. A crucial feature of UARTs for a relatively simple unit of this type is that they can be programed via link-wires connected to control inputs, and they do not have to be set up under software control.

UART

The "raw" output of a UART is not MIDI compatible, but a couple of simple inverter/driver stages are all that is needed in order to provide two outputs having the required five milliamps of drive. A clock oscillator sets the baud rate at the correct figure.

A UART provides a parallel to serial conversion, and will provide any standard word format. For MIDI operation a word format of one start bit, eight data bits, no parity, and one stop bit is required. This format is programmed into the device simply by taking five control inputs to the correct logic states.

In order to transmit the program change messages it is merely necessary to feed the first byte onto the data bus, and then supply a brief pulse to a control input of the UART to initiate transmission of this byte. The second byte (the new program number) is then fed to the data bus, and the control input of the UART is then pulsed again. The data byte is provided by a binary counter which must be incremented each time the unit is activated, so that the program number is automatically advanced each time the unit is operated.

The UART's parallel input bus is fed from two octal tristate buffers. A tristate output is one that can have the usual high or low output levels, but can additionally take up a third state where it is deactivated and provides a high output impedance. In this case the two bufers are controlled by the Q and not Q ($\overline{Q}$) outputs of a divide-by-two flip/flop, and only one or the other will be activated and drive the input data bus of the UART.

CHANGE CODE

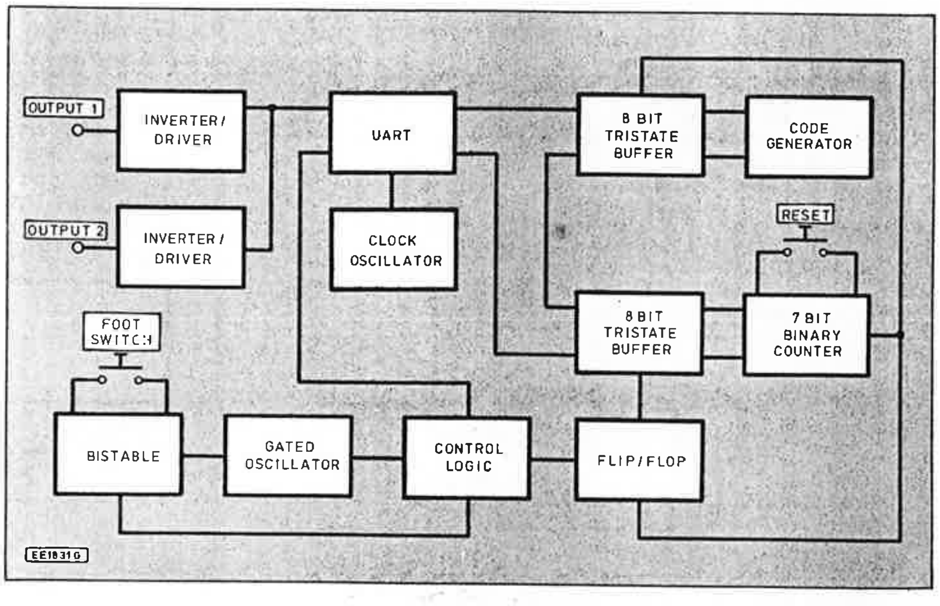
Initially it is the upper of the two buffers that is activated, and this feeds the output from a simple code generator circuit through to the UART. The code generator simply produces 1100 (in binary) as the most significant nibble, and this is the program change code. The least significant nibble can be varied from 0000 to 1111 (again in binary) by means of a switch circuit, and this sets the MIDI channel number.

When operated, the foot switch sets a bistable circuit. This in turn activates a gated oscillator which drives a control logic circuit, and via this, the flip/flop. The pulses from the oscillator and control logic blocks first activate the UART so that the header byte is transmitted. A subsequent pulse clocks the flip/flop to its alternative output states so that the output from the second buffer is fed through to the UART. This buffer is fed from a binary counter, and as explained previously, the most significant bit for data bytes are always set to logic 0. This counter is consequently a seven bit type, and not an eight bit counter.

The output of the counter is set to zero at switch-on, and it can be manually reset to zero at any time using the reset switch. However, the counter is incremented as the second buffer is activated, and it is therefore at 1 when the first data byte is fed through to the UART. This is correct in that the controlled equipment would normally be started at program number 0, and the pedal should generate a change to program 1 when it is first operated.

The next pulse from the control logic circuit causes the data byte to be transmitted by the UART, and the next one resets the bistable. This switches off the oscillator, and returns the unit to the stand-by mode. It remains in this state until the foot switch is operated again. The two bytes are then

Fig. 1. The MIDI Pedal block diagram



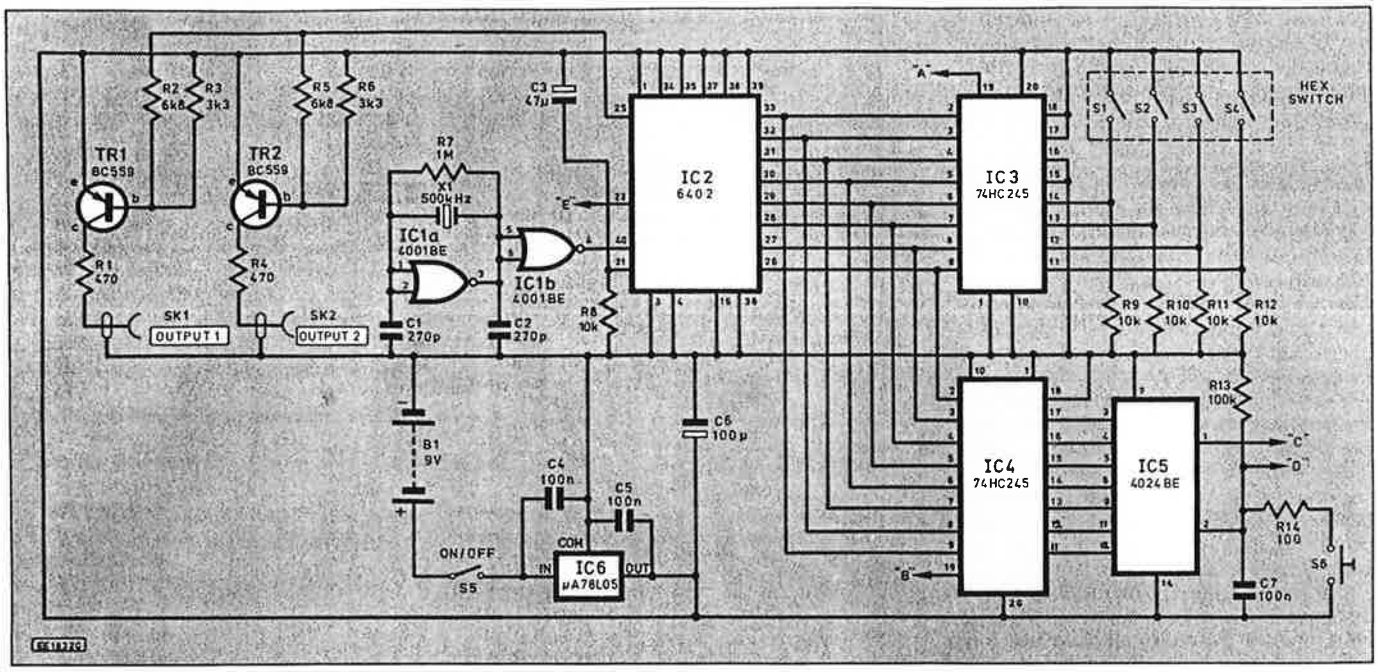


Fig. 2. The circuit for the transmitter section of the unit

transmitted once more, but with the counter being advanced by one prior to the data byte being sent. The unit thus provides the desired action, with a series of automatically incremented program change messages being transmitted.

TRANSMITTER CIRCUIT

Fig. 2 shows the circuit diagram for the section of the unit that generates and transmits the header and data bytes. IC2 is the UART, and this is the industry standard 6402 type. This is supplied with a positive reset pulse at switch-on by C3 and R8. IC1 is a quad 2 input NOR gate, but IC1a and IC1b are wired as inverters. They are used as the clock oscillator and a buffer stage respectively. The clock frequency is set at 500kHz by ceramic resonator X1. IC2 requires a clock signal at sixteen times the transmission baud rate, and this gives the desired 31.25 kilobaud output.

The serial output signal from IC2 drives two common emitter switches (TR1 and TR2) having open collector outputs. The l.e.d.s in the opto-isolators at the inputs of the controlled equipment act as the collector loads for TR1 and TR2, with R1 and R4 limiting the drive current to a suitable figure.

IC3 and IC4 are the octal tristate buffers, and are in fact octal transceivers. However, in this application they are permanently wired in the "receive" mode and are only used as buffers. The four most significant bits of IC3 are tied to the appropriate logic levels to generate the program change code, while the least significant bits are controlled by a hex switch (S1 to S4). Note that if the unit is only required to transmit on MIDI channel number 1, the hex switch can be omitted (but R9 to R12 should still be included, or replaced by link wires).

Hex switches are designed for "on-board" mounting, and this is not a panel mounted control. In the unlikely event that frequent channel changes will be required, probably the best option would be to use miniature toggle switches for S1 and S4, but the channel numbers would then have to be entered in binary form.

IC5 is the seven bit binary counter, and it is reset at switch-on by C7 and R13. It can be manully reset using S6. These signals are also used to reset the flip/flop incidentally.

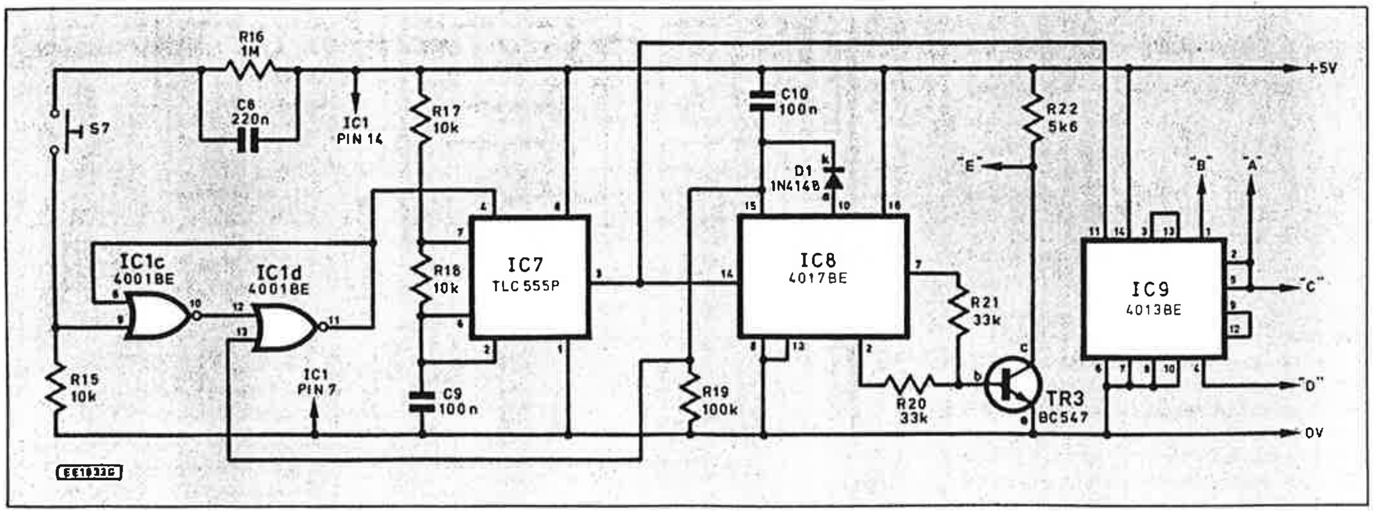
SUPPLY

The circuit requires a reasonably stable five volt supply, and this is obtained from a nine volt battery via monolithic voltage regulator IC6. Battery operation is made feasible by the use of CMOS integrated circuits throughout the design, including the 6402 which has a remarkably low current consumption for such a complex device. The total current consumption of the circuit is only about six milliamps or so.

CONTROL CIRCUIT

The circuit diagram for the control circuitry is shown in Fig. 3. The bistable is formed from the two remaining gates of

Fig. 3. The control logic circuits



IC1. S7 is the foot switch, and it is "debounced" by R16 and C8. This "debouncing" is essential if multiple triggering of the unit is to be avoided. A simple 555 astable circuit based on IC7 provides the gate oscillator stage.

IC8 forms the basis of the control logic stage, and this is a CMOS 4017BE one of ten decoder. It is reset at switch-on by C10 and R19, and it is reset each time output "4" (pin 10) goes high, due to the coupling from this pin to the reset terminal via D1. This effectively relegates IC8 to a one of four decoder. Actually only outputs "1" and "3" are used, and these provide the pulses that initiate the transmission of data from the UART.

Transistor TR3 acts as a simple gate and inverter stage that mixes the two output signals and converts the positive output pulses from IC8 into the negative types required to drive the UART properly. When IC8 resets itself after four oscillator cycles, it also resets the bistable and switches off the oscillator.

IC9 is the divide by two flip/flop circuit, and this is actually a CMOS 4013BE dual D type flip/flop with both sections wired as divide by two stages and connected in series. The additional divider stage is needed because the control logic circuit operates on cycles of four oscillator pulses rather than on two pulse cycles. The extra divider stage keeps the control logic and flip/flop circuits properly synchronised.

CONSTRUCTION

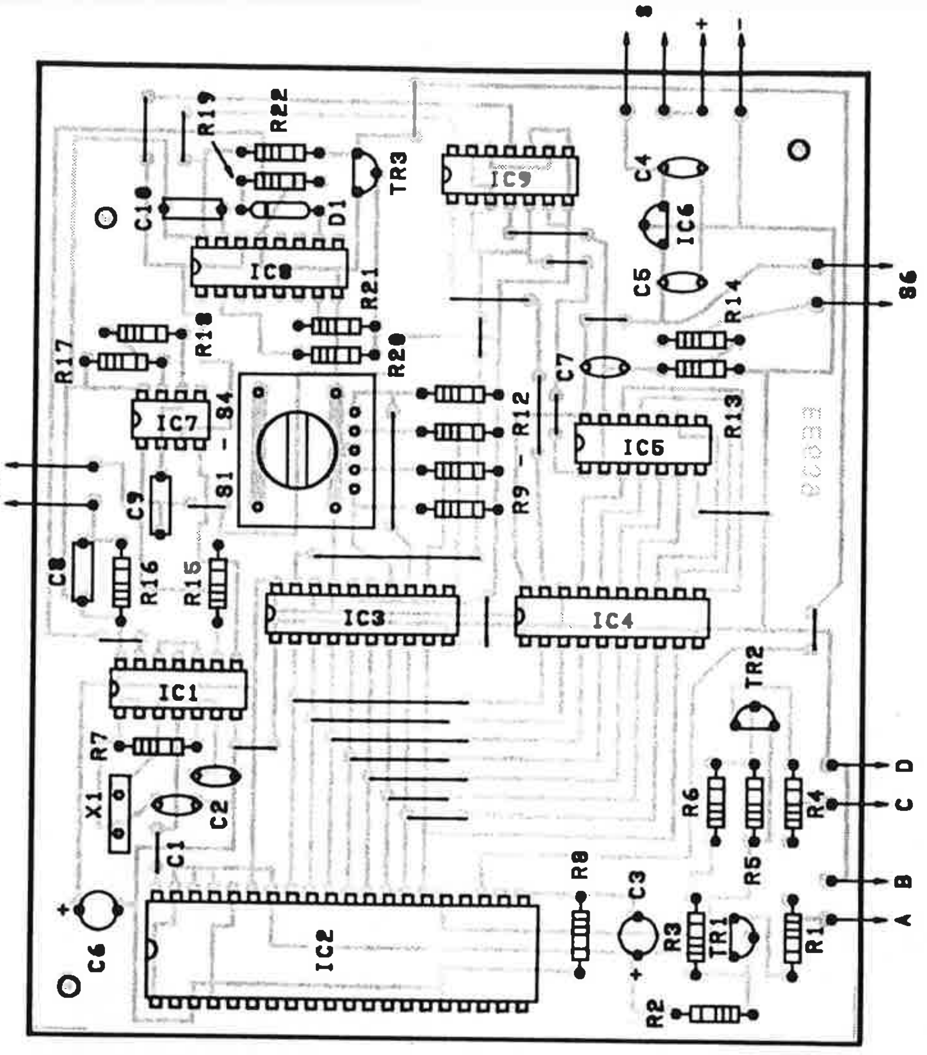
Details of the printed circuit board are provided in Fig. 4. The first point to note here is that all the d.i.l. integrated circuits are CMOS types, and therefore require the normal anti-static handling precautions to be observed. In particular, they should all be mounted in integrated circuit holders. Take special care with the 6402 UART which does not rank as a particularly cheap component. Do not fit the integrated circuits into their holders until the unit is in all other respects finished, and handle them as little as possible.

The board is a single-sided type but a number of link wires are required. These can be made from 22 s.w.g. tinned copper wire. Provided they are kept reasonably taut it should not be necessary to insulate any of them with sleeving. At this stage only pins are fitted to the board at the point where it will eventually be connected to the off-board components.

Provided the specified hex switch is used it should fit onto the board without too much difficulty. Other types may have a different pin arrangement though. Also, the circuit assumes that a switch is closed when it must generate a logic 1 level, but some hex switches go open circuit when they must generate a logic 1 output signal. Use of the specified switch is strongly recommended.

The prototype is housed in an aluminium instrument case which has approximate outside dimensions of 200 by 150 by 50 millimetres, and this comfortably accommodates everything including the batteries. The latter are six HP7 cells fitted in a plastic holder, and these couple to the unit via an ordinary PP3 style battery connector.

The prototype is built to operate with an external foot switch which connects to the main unit by way of a miniature jack socket mounted on the rear panel. Obviously it can be built with a heavy duty non-locking push button switch fitted on the top panel if preferred, and a sloping-front style case



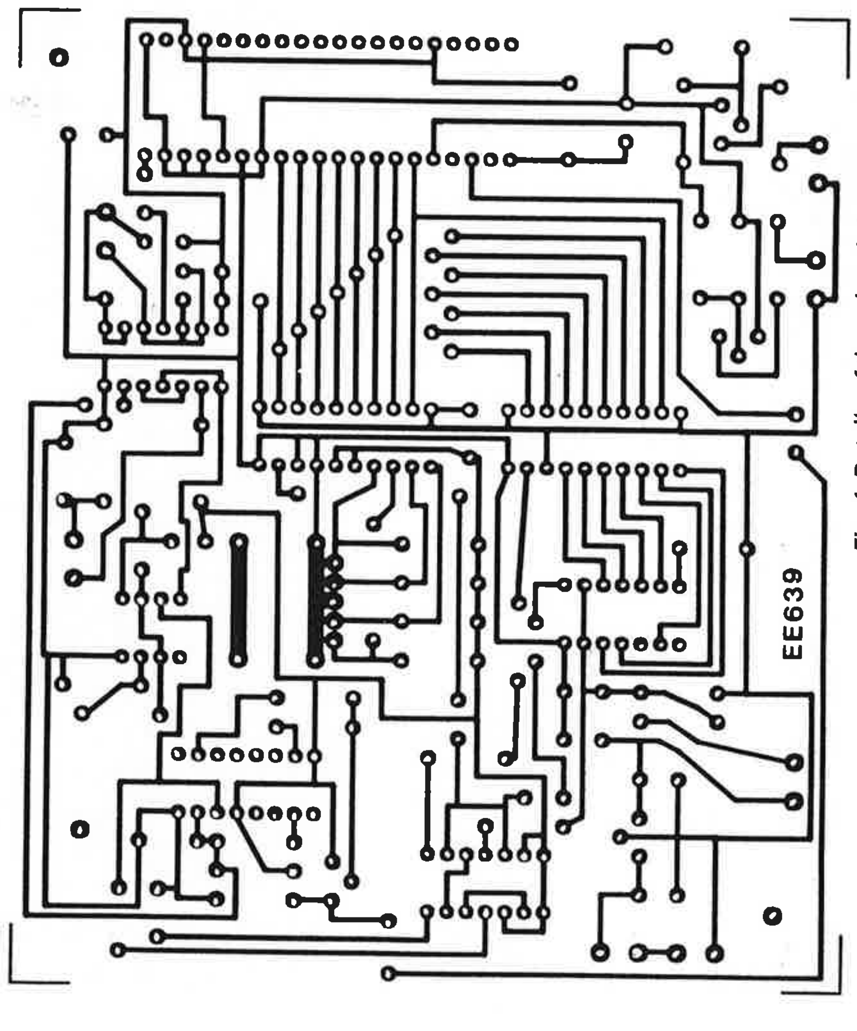


Fig. 4. Details of the printed circuit board.

COMPONENTS

Shop Talk
See page 193

Resistors

R1, R4	470 (2 off)
R2, R5	6k8 (2 off)
R3, R6	3k3 (2 off)
R7, R16	1M (2 off)
R8 to R12, R15, R17, R18	10k (8 off)
R13, R19	100k (2 off)
R14	100
R20, R21	33k (2 off)
R22	5k6

All 0.25W 5% tolerance

Capacitors

C1, C2	270p ceramic plate (2 off)
C3	47μ radial elect 16V
C4, C5, C7	100n ceramic (3 off)
C6	100μ radial elect 10V
C8	220n polyester
C9, C10	100n polyester (2 off)

Semiconductors

IC1	4001BE CMOS quad 2 input NOR
IC2	6402 UART
IC3, IC4	74HC245 octal transceiver (2 off)
IC5	4024BE 7 bit ripple counter
IC6	μA78L05 100mA 5V regulator
IC7	TLC555P low power timer
IC8	4017BE CMOS 1 of 10 decoder
IC9	4013BE CMOS dual D type flip/flop
D1	1N4148 silicon signal diode
TR1, TR2	BC559 silicon *pnp* (2 off)
TR3	BC547 silicon *npn*

Switches

S1 to S4	Hex switch
S5	S.P.S.T. sub-min. toggle
S6	Push to make, non-locking type
S7	Push button (see text)

Miscellaneous

B1	9 volt (six HP7s in plastic holder)
SK1, SK2	5 way 180 degree DIN socket (2 off)
X1	500kHz ceramic resonator

Case about 200×150×50 millimetres; printed circuit board available from the *EE PCB Service*, order code EE639; 8 pin d.i.l. i.c. holder; 14 pin d.i.l. i.c. holder (3 off); 16 pin d.i.l. i.c. holder; 20 pin d.i.l. i.c. holder (2 off); 40 pin d.i.l. i.c. holder; battery connector (PP3 style), pins, wire, solder, etc.

Approx. cost Guidance only **£30** inc. case

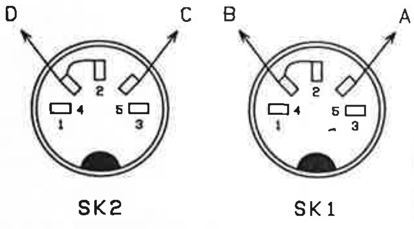


Fig. 5. The connections to SK1 and SK2.

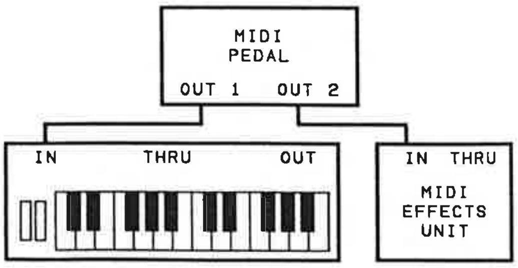


Fig. 6. Suggested MIDI wiring for a system having one instrument plus a MIDI effects unit

might then represent the best form of housing for this project. Both the case and the switch must be heavy duty types if a built-in foot switch is used.

Whichever form of construction is selected, the printed circuit board is mounted on the base panel of the case on stand-offs, and the on/off switch, reset switch, and output sockets are mounted on the front panel. Five way 180 degree DIN sockets were not a random choice for SK1 and SK2 —these are the standard MIDI connectors. Fig. 5 shows the correct method of wiring these to the printed circuit board. The unit is completed by wiring the battery clip, reset switch, and on/off switch to the board, and then finally connecting either the foot switch or its socket on the rear panel.

TESTING AND USE

To test the pedal, simply connect one of the output sockets to the MIDI IN socket of any item of equipment that responds to MIDI program change messages. Ideally you should use something like a synthesiser that shows the current programme number on its display, or you may have a computer plus MIDI interface and a utility program that displays received data.

Repeatedly activating S7 should result in the program switching the 1 first, then 2, and so on. If you make your own connecting cable, pins 4 and 5 on one plug connect to the same pins on the other plug. They are not cross connected (as in some DIN audio leads). Pin 2 connects to the cable's screen.

There are a few points that need to be borne in mind when testing and using the unit. First of all, bear in mind that if the controlled equipment is set to an "omni off" mode, it will almost certainly not respond to MIDI program change messages unless the MIDI pedal and the controlled equipment are set to the same MIDI channel.

Many pieces of MIDI equipment can be set to ignore program change instructions, and obviously, where appropriate, the controlled equipment must be set to respond to program change messages.

ANOMALIES

Confusion often arises with MIDI due to anomalies in the numbering of programs and channels. The hex switch will almost certainly be numbered from 0 to 15, which is the range of binary values it produces. On the other hand, the convention is for MIDI channels to be numbered from 1 to 16. Accordingly, if you wish to set the panel to (say) MIDI channel number 9, the hex switch should be set to position "8". In other words you must set the switch to one less than the desired MIDI channel.

Things are less straightforward with MIDI program numbers. The actual range of numbers used in the messages is from 0 to 127, and this is the program numbering range used by some manufacturers. However, others use 1 to 128, and some use a totally different system. For instance, I have a Casio synthesiser which provides manual program selection via two sets of push button switches marked A to H, and 1 to 8. The programs are therefore numbered from A1 to H8!

The equipment manuals should make it clear what method of numbering is used, and provide a conversion table where an unusual method of numbering is in use. Bear in mind that most pieces of equipment do not use the full range of 128 programs. Sending an out-of-range program number will normally just result in it being ignored by the equipment, rather than causing anything catastrophic.

CONNECTING

The normal method of connecting two or more items of MIDI equipment together is to connect the THRU socket of one unit to the IN socket of the next, and so on, building up a chain of connections as long as required. In practice things do not necessarily work out quite as easy as this.

Many MIDI instruments, especially the keyboard types, lack a THRU socket. Also, the chain system can produce poor reliability due to the signal being slightly degraded as it passes through each unit. The extra output on the pedal unit can therefore be very useful, and it is probably worthwhile using it, even where the chain system of connection could be used. For example, with a synthesiser and effects unit that are both to be controlled from the pedal the method of connection shown in Fig. 6 would be the best one to use. □