MIDI HORN
The Article

Alternative Performer Interfaces for MIDI synthesizers

by

Gary Lee Nelson, Professor
and
John Talbert, Music Engineer

TIMARA Department
Conservatory of Music
Oberlin College

Oberlin Conservatory
TIMARA Department

Alternative Performance Devices

How would you like to play a horn that commands a full orchestra of sounds with a single breath? Or perhaps you would prefer painting sound in the air with a wave of your hands or dancing across the room with every movement of your body echoed by some nuance in the music. These alternative performance techniques can be based on a common technology that is already available. By the end of this article you will have the means to create instruments tailored to your vision of composition and performance.

New Instruments

Many new instruments have reached the experimental stage and some are already in commercial production. Michel Waisvisz's "Hands", Airdrums by Palmtree Instruments, a Midi Saxophone by Artisyn are freeing electronic music from the sole domain of the keyboard player.

We will introduce some techniques that can form the basis of other new instruments. Sounds will be produced by your own MIDI synthesizers. How you control those sounds depends on your implementation of the system outlined in Figure 1. You create your own performance controller and then interface it with a microcomputer. The microcomputer translates your actions into MIDI commands and passes those commands on to the synthesizers. Let's take a look at the system one part at a time.

Let me introduce you to a scheme which can form the basis for creating amazingly diverse new instruments. It is really quite simple and at the same time extremely flexible. Figure 1 shows the gist of it. The actual sounds are produced by your own Midi-equipped synthesizers. How you get at those sounds, however, is covered by the device shown in the figure. You create your own performance device and then interface it to a microcomputer which translates your actions into midi sound commands. Let's take a look at it one part at a time.

Performance Devices

The "performance device" of our new instrument can be the most creative part of the system. To the audience this <u>is</u> the instrument. For us, it is a collection of switches and transducers tailored to our particular performance preferences. You can use push buttons, toggle switches, slide controls, turn pots, foot pedals, ribbons, pressure transducers, and strain gauges. You can use detectors that respond to light, ultrasonic signals, and infrared waves. The many ways of connecting these controls to the performer suggest a broad range of methods for playing synthesizers. Some of these devices can be very easy to implement - the switches and potentiometers, for instance. Some of the other transducers will take a bit more electronics. We will have a closer look at some specific cases later when we talk about our own performance device, the MIDI Horn.

The Interface

The analog to digital conversion (ADC) section of our system translates the instrument's switch and transducer outputs to a form that the microcomputer can understand. The switches can be connected directly to the microcomputer's I/O lines. The open or closed connection in a switch is read as a single high or low digital bit by the microcomputer. For the other devices we add circuitry to produce continuous changes in voltage over a range of 0 to 5 volts. We send these control voltages to an ADC that transforms the voltage into discrete digital values in a range of 0 to 255. These digital number are interpreted as performance actions by the program running in the microcomputer.

The microcomputer continually scans the changing outputs from the performance device and converts them into MIDI commands. It translates the gestures of the performer into musical responses. The precise nature of the responses depend on how you program the microcomputer. Programs can be changed easily to make instruments grow and change as our performance techniques develop. Program changes might be made to fine tune the microcomputer's response to particular controllers or a new program could transform the device into an entirely different instrument. Programs may be structured so that the instrument changes characteristics during a performance. Through the following discussion, we must remember that we are undertaking instrument design at a level that is more fundamental than programming a voice on a synthesizer. We are reconsidering the assumptions and prejudices about the relationship between performers and their musical tools.

THE MIDI HORN

Wind players have, to a great extent, been left out of the MIDI revolution. The simple addition of a breath controller to a keyboard synthesizer does not solve the problem. Wind players become proficient at synchronizing tongue, air flow, and finger muscles into a distinctive set of performance gestures. Let's look at our design for an instrument that allows brass players to play MIDI synthesizers with idiomatic modes of expression. We call it the MIDI Horn. (See Figure 2).

On one end of the horn is a breath controller. This device contains a pressure transducer that changes breath pressure into a voltage which can be read by the microcomputer. Brass players buzz their lips at different frequencies to accomplish changes in register and ranges of at least three octaves are common. However, the MIDI Horn does not respond to buzzing but only to air pressure. Also, unlike brass instruments, the tube is closed so that no air actually flows through it. If the performer is more comfortable with a brass mouthpiece, one can be attached to the breath controller tubing either directly, or with a short piece of shrink wrap tubing. The performer's favorite mouthpiece can serve to focus the air flow but the size doesn't matter.

The breath controller has been programmed to perform its normal MIDI function (Breath control #2) but it has also been given the job of determining when notes start and stop. When the breath pressure rises above a certain threshold a Midi Note-On command is sent The note remains on until the breath pressure drops below the threshold again, causing a Midi Note-Off to be sent. The initial volume of the note is determined by how fast the pressure is changing when the threshold is passed. The actual Note value is determined by what keys, on the top of the instrument, are pressed when the threshold is passed. The threshold can be adjusted by a small potentiometer on the horn, and also on the microcomputer circuit board.

After a Note-On event, the Horn is programmed to continue watching for changes in pressure at the transducer. Whenever a change is detected, the new pressure value is sent out in the form of MIDI Controller #2 data. This continues until the pressure drops below the threshold causing a Midi Note-Off to be sent. Most synthesizers can be setup to use Midi Control#2 data to change such sound parameters as volume and/or modulation depth - check your synthesizer manual.

On the top of the instrument are eight push buttons. Computer keyboard switches with a fairly deep stroke are used because they approximate the feel of brass instrument valves and because they are designed for extended use. The top group of four keys has been programmed to simulate brass fingering for twelve notes in an octave. Trumpet players may be a bit disturbed but most other brass players are used to four valves. Each key lowers the pitch of the instrument by a number of semitones as follows:

first valve	2 semitones
second valve	1 semitone
third valve	3 semitones
fourth valve	5 semitones

These valves used singly and in combination are suitable for producing a chromatic scale of 12 different pitches. The usual alternate fingerings are preserved for the convenience of the brass performer.

One additional function of the top four switches is to set the MIDI Channel number. The position of these 4 keys is read once when the microcomputer is powered on or the Reset button is pressed. The value read is then used for all subsequent MIDI data.

The next set of 3 keys is programmed to control register by allowing the selection of eight different octaves. The fingering pattern is identical to that of the first three valves in the group of four. The interval that is subtracted is an octave rather than a semitone. The third valve by itself is an alternate fingering for the 1-2 combination and has been appropriated to reach the lowest octave. Several players who have tried this scheme have adjusted very quickly to these small deviations from normal brass fingering technique.

One final key at the bottom is operated by the pinkie finger and has been programmed to control MIDI voice changes. When this key is pressed the upper seven keys are interpreted in a binary pattern and entered as a MIDI Program number (0 to 127). The performer can thus change timbre without removing the hands from the instrument.

Additional controllers on the back of the Horn are operated by the player's thumbs. These consist of 8 push-on/push-off switches and two joysticks. Changes in the back switches are sent out as MIDI Control numbers 84 to 91. Changes in the joysticks are sent out as MIDI Control numbers 16 to 19, configured as follows:

top joystick,	up-down	Control #16
top joystick,	left-right	Control #17
bottom joystick,	up-down	Control #18
bottom joystick,	left-right	Control #19

As with the MIDI Breath Control, these controllers can be set up at the synthesizer to control various synthesizer parameters.

The Horn Circuits

The wiring for the Horn is not too complicated. The front 8 keys are momentary switches that are normally open. One side of each switch is grounded while the other side goes to Microcomputer Port F at the interface box. The back 8 push-on/push-off keys are wired similarly and go to Microcomputer Port G at the interface box.

Each joystick consists of two pots wired to ground and a conditioned voltage supply. The pot outputs go from zero to about 4 volts as the joystick is moved. The 4 joystick continuous voltage outputs are sent to an Analog to Digital Converter at the interface box.

The Series PX136 transducer used for breath pressure is made by Omega Engineering, Inc. An opamp circuit with a sensitivity adjustment pot amplifies the pressure signal before sending it off to an additional amplifier and, eventually, an Analog to Digital Converter at the interface box.

Cabling between the horn and the interface box containing the microprocessor uses a standard computer cable with male D25 plugs at each end. A diagram of the cable shows the pin-outs for the 16 switch outputs, the 5 continuous controller outputs, and the 5 volt power and ground.

The Interface Circuits

The Interface box houses the microprocessor that converts the signals from the performance devices to MIDI output data. The front panel of the interface box has a power on/off switch, a power indicator lamp, a D25 plug for connection to the Horn or similar device, and a Reset push-button for restarting the main program. (It also has some features used in the program development process: a couple switches for loading programs into an EEProm, and a D25 plug wired for RS232 serial communication between the interface microcomputer and a terminal.) The rear panel has a jack for an AC power transformer, two MIDI output jacks, and a non-implemented MIDI input jack.

The rear panel of the interface box also has jacks for 4 controller devices which are in addition to the MIDI Horn controls. There are 2 mono 1/4" phone jacks for switch type devices and two stereo 1/4" jacks for continuous controller type devices. They are programmed to send out MIDI Control data as follows:

Top Switch Jack	Control #64
Bottom Switch Jack	Control #65
Top Pedal Jack	Control #1
Bottom Switch Jack	Control #4

The 2 switch jacks are Mono 1/4" phone. Any action which connects or disconnects the tip from the sleeve in the jack will cause a MIDI control signal to be generated.

The 2 pedal jacks are Stereo 1/4" phone. These are meant for continuous controller devices that put out a voltage variable between 0 and 5 volts. This variable voltage from the device must be connected to the **ring** of the stereo jack. The **tip** of the stereo jack provides 5 volts and the **sleeve** provides ground from the interface power supply for use by the device. See the extra sheets for ideas on how to build simple light and pressure sensors.

A total of 18 switch signals and 7 continuous controller signals go into the Interface box for processing.

The connection for the switches is fairly simple: Each of the 18 switch lines is connected directly to one line of an input port on the microprocessor chip. A pull up resistor tied to 5 volts exists at each of these input port lines within the microprocessor so that normally open keys rest at a high of 5 volts. When a key is depressed its line is directly connected to ground forcing it to 0 volts. The state of the switches can be read at any time by the processor. The program then determines what action is taken (i.e. what MIDI data is sent) whenever the state of a switch changes.

The interface required for the 7 continuous controller signals is more involved as seen in the Midi Horn ADC circuit diagram. Each continuous controller signal is first filtered to take out any changes or glitches in the signal faster than about a millisecond. After an opamp buffer it enters an Analog to Digital Converter (ADC 0809) which converts each of the 7 continuous controller signals into an 8 bit digital word with values from 0 to 255. The ADC runs continuously converting one signal at a time and letting the microprocessor know when a conversion is complete so that it can pick up the results and store it. Each conversion takes about a tenth of a millisecond. This results in each signal being sampled about once a millisecond.

The MIDI signals are generated by a 6850 serial interface chip as shown in the MIDI Horn / Midi Interface circuit diagram. The microprocessor feeds the 6850 with MIDI data as dictated by the program. The 6850 chip then outputs the data in serial form at the MIDI baud rate (31.25K).

Finally, the bulk of the circuitry in the interface box is the microcomputer - an NMIX-0012 single board microcomputer from New Micros Inc., Texas using a R65F12 microprocessor.

The Program

At this point in our description the Midi Horn we have performance signals ready to be read by a microcomputer and we have a synthesizer waiting for midi commands. The only thing needed now is a program to make the link. Our MIDI Horn program is structured as a loop that reads signals from the performance device and translates them into MIDI commands. We decided to use Forth - a language that is fast and easy to work with. Forth was originally developed for real time processing and we found it to be ideal for our time-critical program loop.

The Rockwell 65F12 is incorporated into a single board computer marketed by New Micros Inc. The 65F12 chip is based on the 6502 processor used in the old Commodore, Atari and Apple II microcomputers. The processor includes its own RAM memory and a kernel of Forth subroutines in ROM.

Forth code is written by constructing subroutines, or "words" in Forth jargon. Each construction you see in the listing that is delimited by a colon at the start and a semicolon at the end is the definition of a Forth word. The name for the defined word immediately follows the colon. Once defined, a word is added to the "dictionary" and can be used as part of subsequent definitions. Even if you don't know Forth the following comments should give you a general idea of the program logic.

The program that runs our instrument is found in the word HORN (see Listing 1) . Horn is composed of words previously defined along with other words that are defined in the Forth Kernel in ROM. The HORN program is automatically run when the interface box is turned on or the reset switch is pressed. The program was developed on this same board using its serial interface connected to a desktop computer running a terminal emulation program. Each word developed from the terminal can be run and tested by typing the word and then a carriage return.

The main program word HORN is made up of many previously defined words which are in turn made up of more basic words. Some of the words require parameters to do their jobs. Others may produce results that they want to communicate to the program. The word ON, for example, requires a key velocity, key number, and MIDI channel number. A note will be turned on in the synthesizer that is connected to your MIDI output when ON is executed. Turning notes off requires a similar action. This brief description of Forth should enable you to follow our description of the program.

The Horn Program

Figure 4 shows a flow diagram for the main program word, HORN. Note the correspondence between the definition of HORN in the listing and this flow diagram. The program is a loop that reads the MIDI Horn signals, checks the bottom key for a possible voice change, and then asks some questions to determine the next program branch. The program first asks whether a note is currently sounding. If the answer is yes we follow the left side of the diagram to check the performer's breath pressure. If the breath pressure drops below the "off threshold", the program sends a MIDI key off command. If the player changes fingering in the same breath, the program will send a MIDI key off command for the note in progress followed by a MIDI key on command for the new note. While a note is sounding, the player's breath pressure is transmitted to MIDI out as a continuous control change for the synthesizer. The other side of the flow diagram is much simpler. If no note is currently sounding the player's breath pressure is checked against an "on threshold". If the breath pressure is above the threshold, the program reads the seven front keys to determine pitch. A note is started with a MIDI key on command.

See the Listing "Horn Program" for the actual Forth subroutines that make up the main HORN program. These subroutines have been compiled and stored in the ROM memory chip labeled as HORN 1.0. Comments are written between parenthesis.

An Application

Professor Gary Nelson has used the MIDI Horn extensively in performances with a system that includes a Macintosh computer and several synthesizers such as the Yamaha TX816 and EMU Proteus modules.

MIDI commands from the horn other pedals connected to the interface box are interpreted by an interactive MAX (by Opcode) program running on the Macintosh. The MAX program recognizes MIDI commands received from the Horn and calls user-written subroutines to handle each MIDI event. Events can be echoed immediately, stored for later use, or transformed according to some compositional plan.

Pressing a key may trigger a prerecorded sequence of notes stored in the Macintosh, or a composition algorithm that computes notes on the fly. Pressing the sustain pedal may throw the sequence into reverse by setting a retrograde flag. Likewise, the portamento pedal can be used for inversion or some other toggle function. The key number fingered on the MIDI Horn can set the bass pitch for transposing the sequence. In short, the MIDI signals coming into the Macintosh are simple stimuli that can be interpreted any way your musical fancy dictates. The possibilities suggested in our discussion of transformations in the Forth computer are greatly increased by the added computational stage of a Macintosh running MAX placed between the MIDI Horn and your orchestra of synthesizers.

Conclusion

We hope that we have inspired you to create your own MIDI performance interface. In computer music, esthetic ideas are shaped by both hardware and software but, most of all, they are shaped by the fantasy of the artist. Composers and performers must take an active role in evolving process to show industry and the commercial world where real progress is to be made.

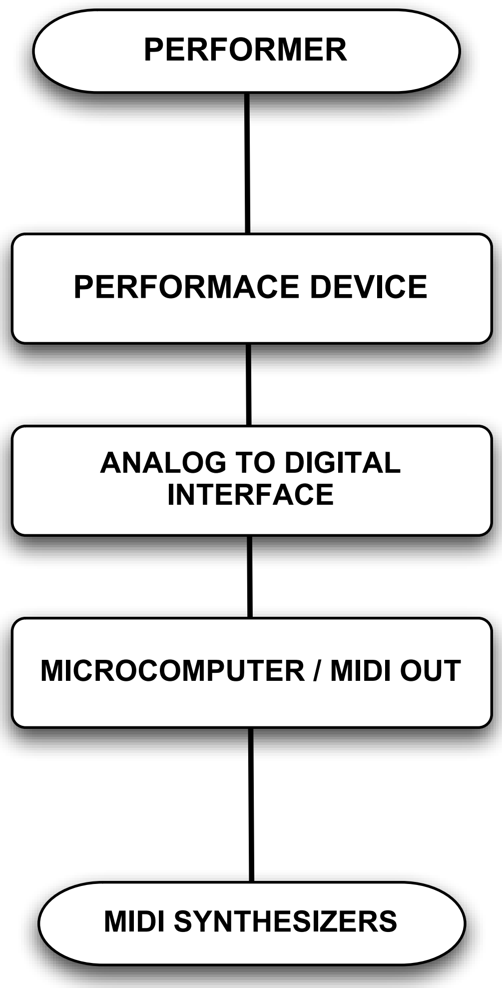


Figure 1. The Structure of a New MIDI Instrument

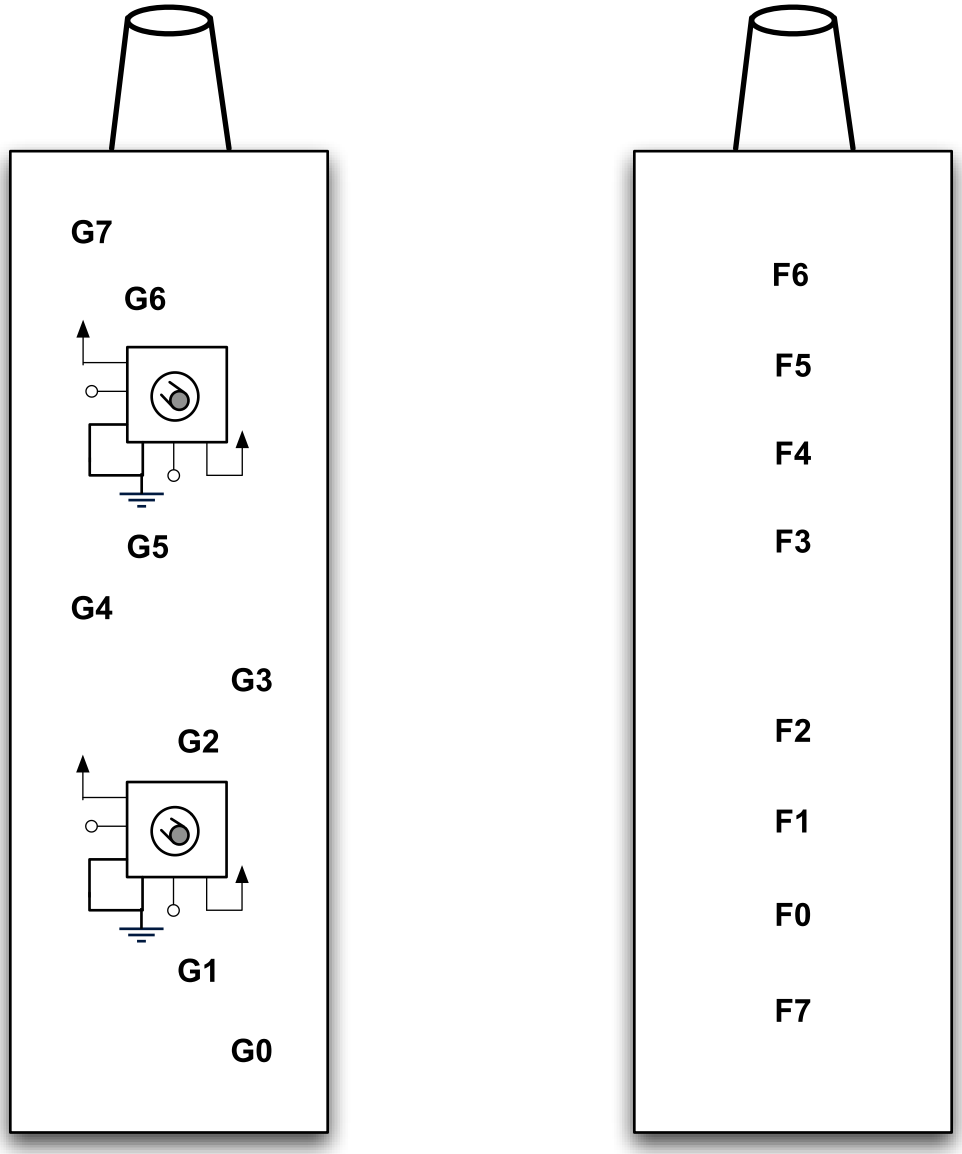


FIGURE 2 - Switches and Joysticks

MIDI Horn
Note Table

Fingering 1 2 3 4	Key Value
– – – –	12
– * – –	11
* – – –	10
* * – –	9
– – * –	9
– * * –	8
* – * –	7
– – – *	7
* * * –	6
– * – *	6
* – – *	5
– – * *	4
* * – *	4
– * * *	3
* – * *	2
* * * *	1

MIDI Horn
Octave Table

Fingering 1 2 3	Key Value
– – –	96
– * –	84
* – –	72
* * –	60
– * *	48
* – *	36
* * *	24
– – *	12

* Button Pressed

Key Number = Note + Octave

FIGURE 3 - Fingerings

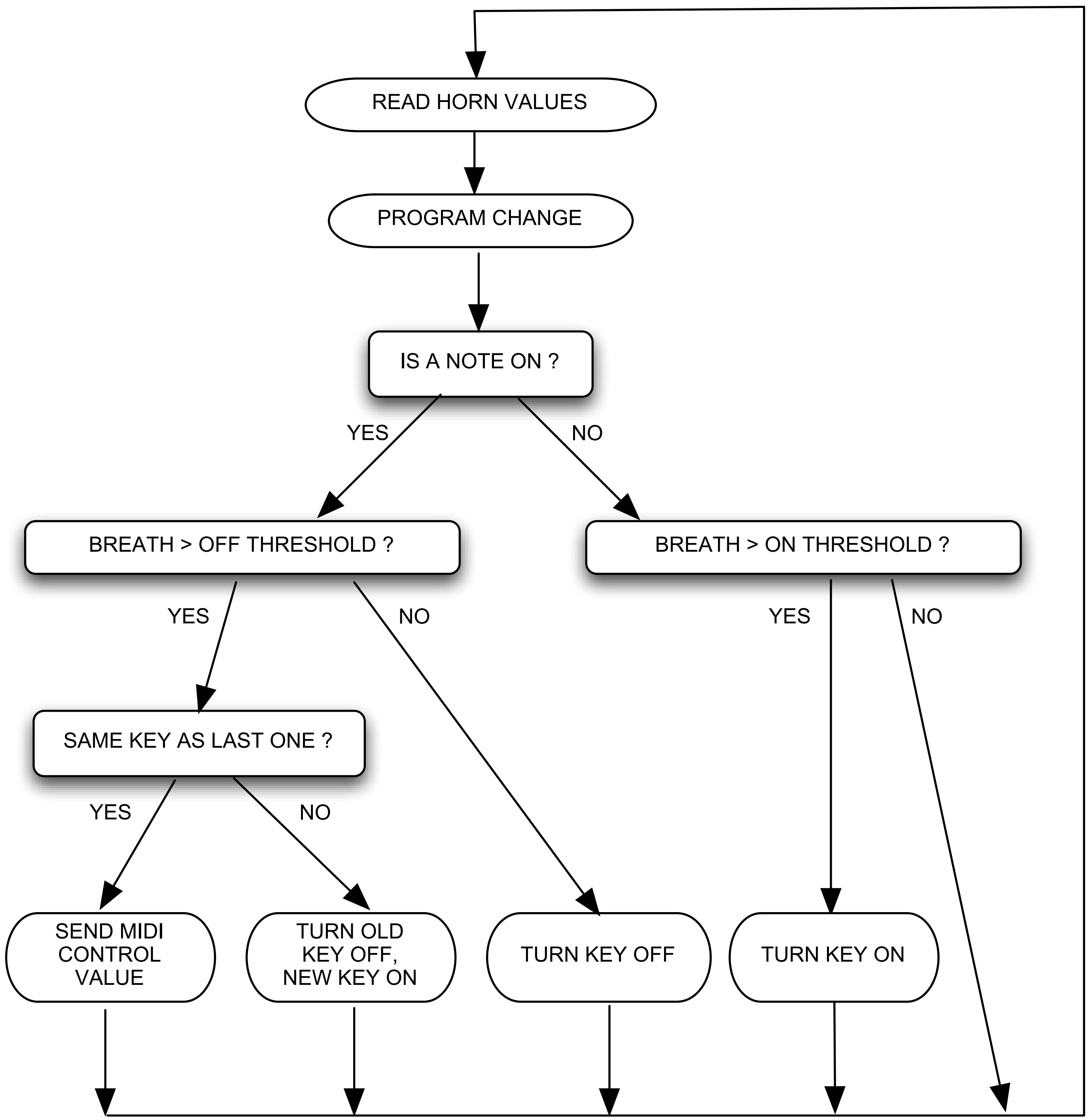
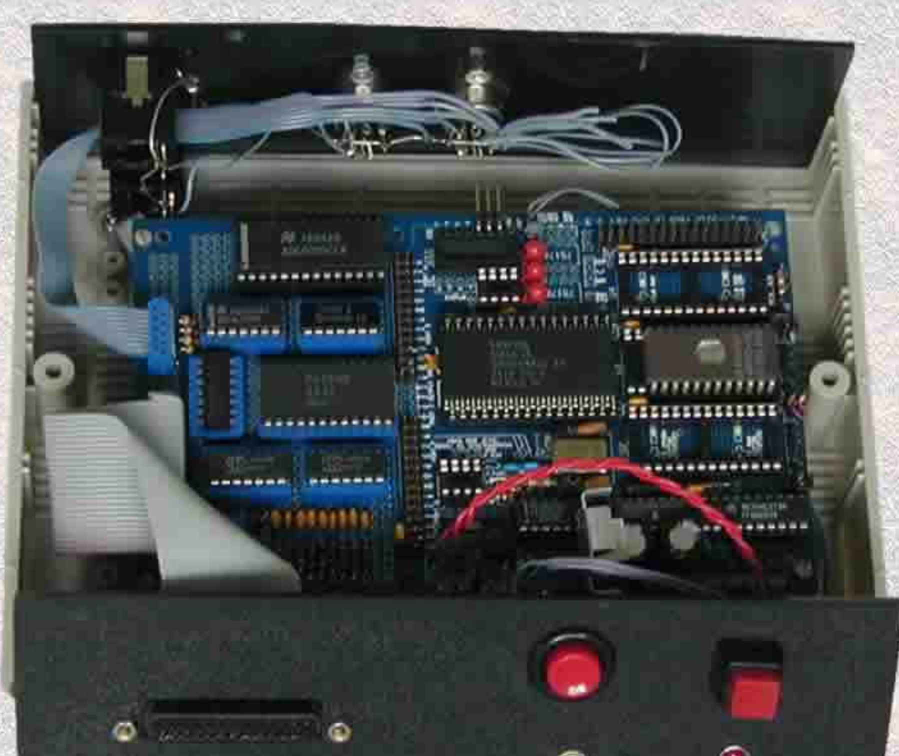


FIGURE 4 - Program Flow Diagram

MIDI HORN
The Circuit



MIDI HORN CABLE
25 Pin D Plug

1	GND
2	ANAL 7, Bottom Joystick, side to side
3	ANAL 2, Top Joystick, side to side
4	--
5	G6
6	G4
7	G2
8	G0
9	F6
10	F4
11	F2
12	F0
13	GND

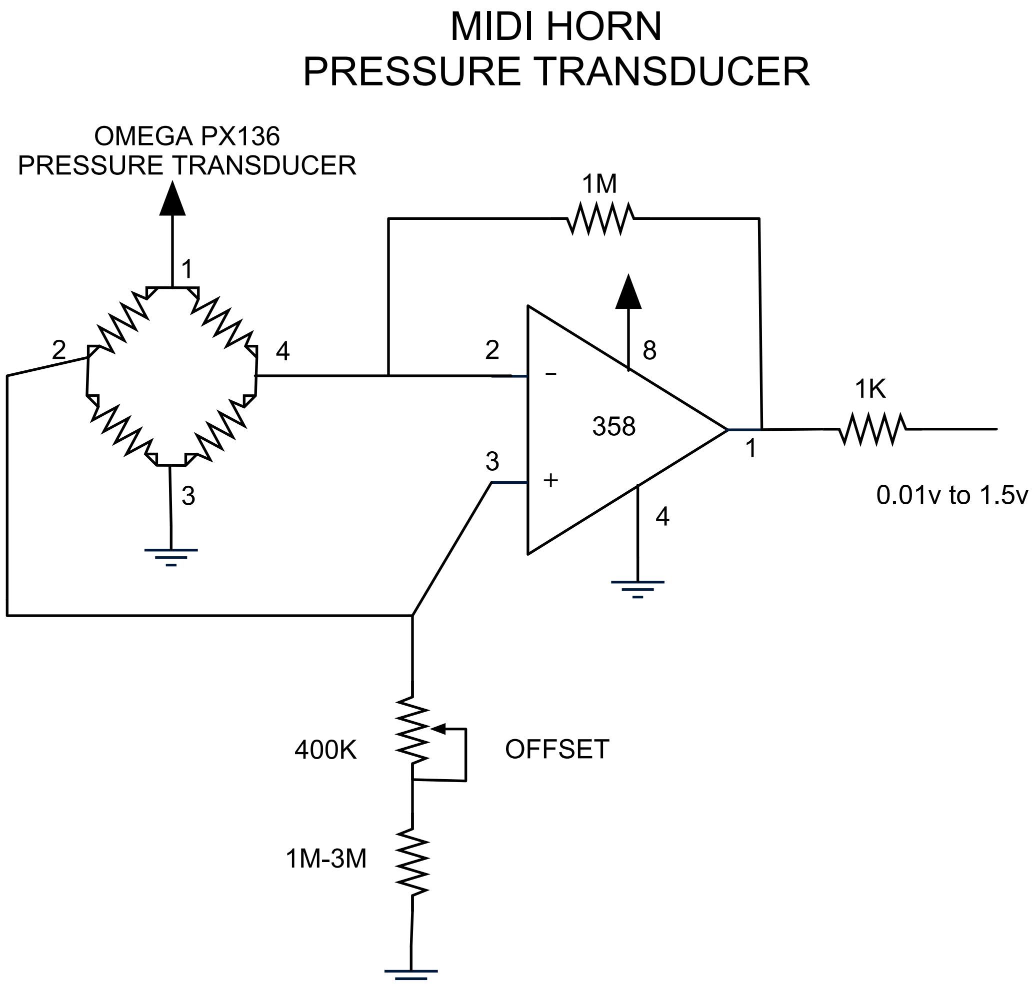
14	ANAL 0, Breath Control
15	ANAL 6, Bottom Joystick, up and down
16	ANAL 1, Top Joystick, up and down
17	G7
18	G5
19	G3
20	G1
21	F7
22	F5
23	F3
24	F1
25	+5v

FIGURE 5 - Horn Cable Connections

MIDI HORN
Controller Memory Addresses

Controller	MIDI Controller #	Computer Address	Computer Device
Analog to Digital Converter (01 xxx0 xxxx)			
Breath Pressure	2 (2h)	0100h	ADC 0
Top Joystick, up/down	16 (10h)	0101h	ADC 1
Top Joystick, left/right	17 (11h)	0102h	ADC 2
Unused		0103h	ADC 3
Continuous Pedal 1	1 (1h)	0104h	ADC 4
Continuous Pedal 2	4 (4h)	0105h	ADC 5
Bottom Joystick, up/down	18 (12h)	0106h	ADC 6
Bottom Joystick, left/right	19 (13h)	0107h	ADC 7
Horn Pushbuttons on Micro Ports			
Front Pushbuttons bottom to top	bit 7 = Program Ld bits 6-3 = Note# bits 2-0 = Octave	Port F0 to F7	Mico Port F
Back PushOn/PushOff bottom to top	84 to 91	Port G0 to G7	Micro Port G
Sustain Pedals	64 and 65	Port B0 and B1	Micro Port B
Serial MIDI 6850 UART Chip (01 xxx1 1xxx)			
MIDI Out	-----	0118h	UART Control Reg. setup with MIDINIT
MIDI Out	-----	0119h	UART Data Load with MLD

FIGURE 6 - Horn Device Addresses



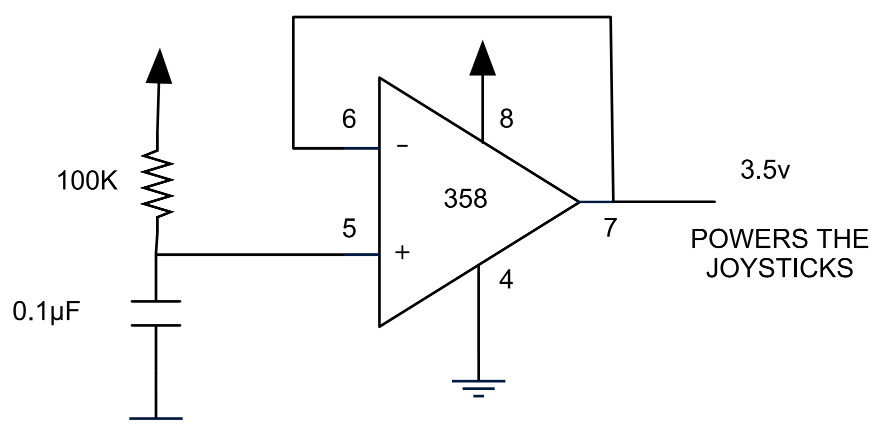


FIGURE 7 - Pressure Transducer

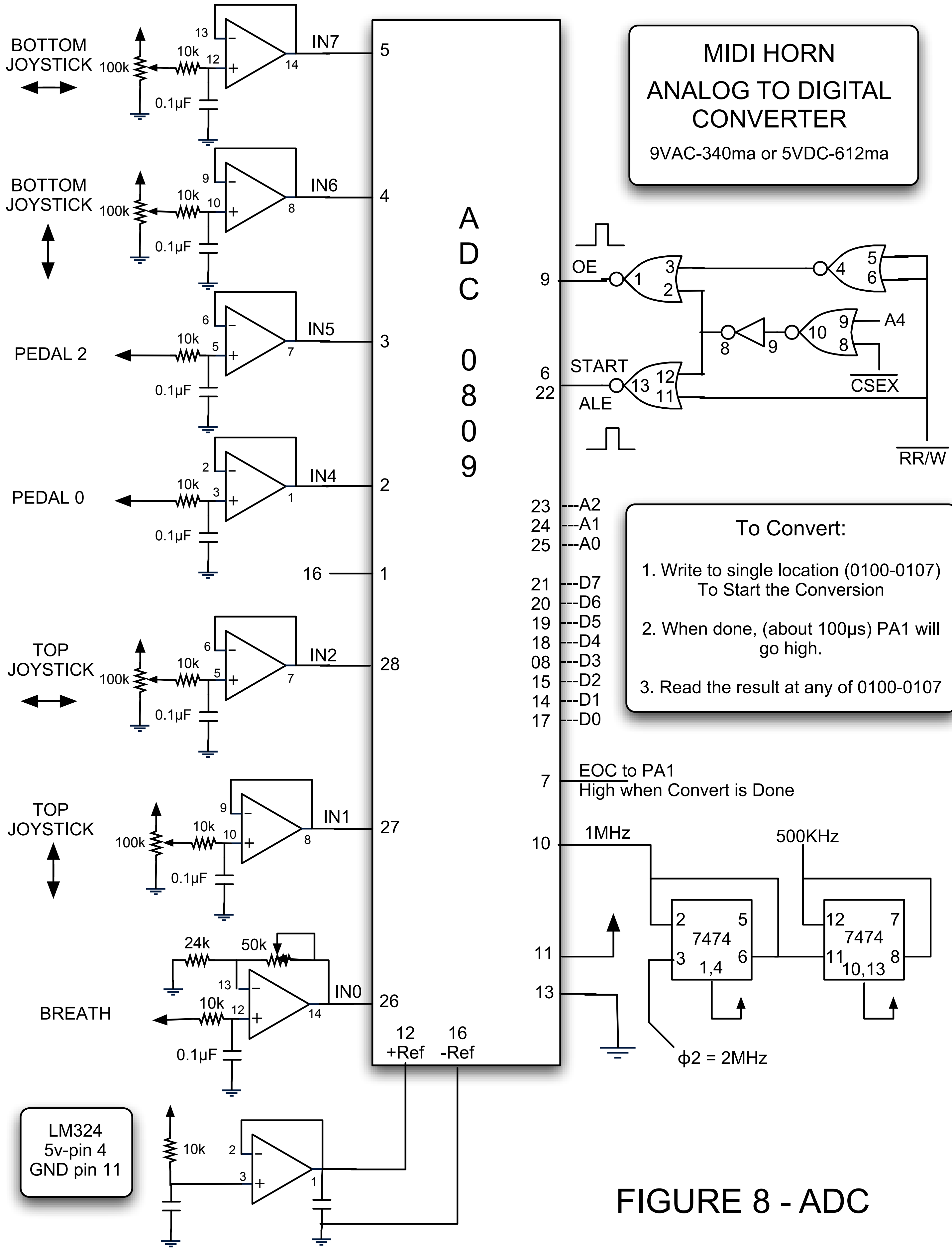

MIDI HORN
ANALOG TO DIGITAL CONVERTER
9VAC-340ma or 5VDC-612ma
BOTTOM JOYSTICK
BOTTOM JOYSTICK
PEDAL 2
PEDAL 0
TOP JOYSTICK
TOP JOYSTICK
BREATH
100k
10k
0.1µF
24k
50k
IN7
IN6
IN5
IN4
IN2
IN1
IN0
A D C 0809
OE
START
ALE
A4
CSEX
RR/W
23 ---A2
24 ---A1
25 ---A0
21 ---D7
20 ---D6
19 ---D5
18 ---D4
08 ---D3
15 ---D2
14 ---D1
17 ---D0
EOC to PA1
High when Convert is Done
1MHz
500KHz
7474
1,4
7474
10,13
ϕ2 = 2MHz
12 +Ref
16 -Ref
LM324
5v-pin 4
GND pin 11
To Convert:
1. Write to single location (0100-0107) To Start the Conversion
2. When done, (about 100µs) PA1 will go high.
3. Read the result at any of 0100-0107

FIGURE 8 - ADC

MIDI HORN
MIDI Interface

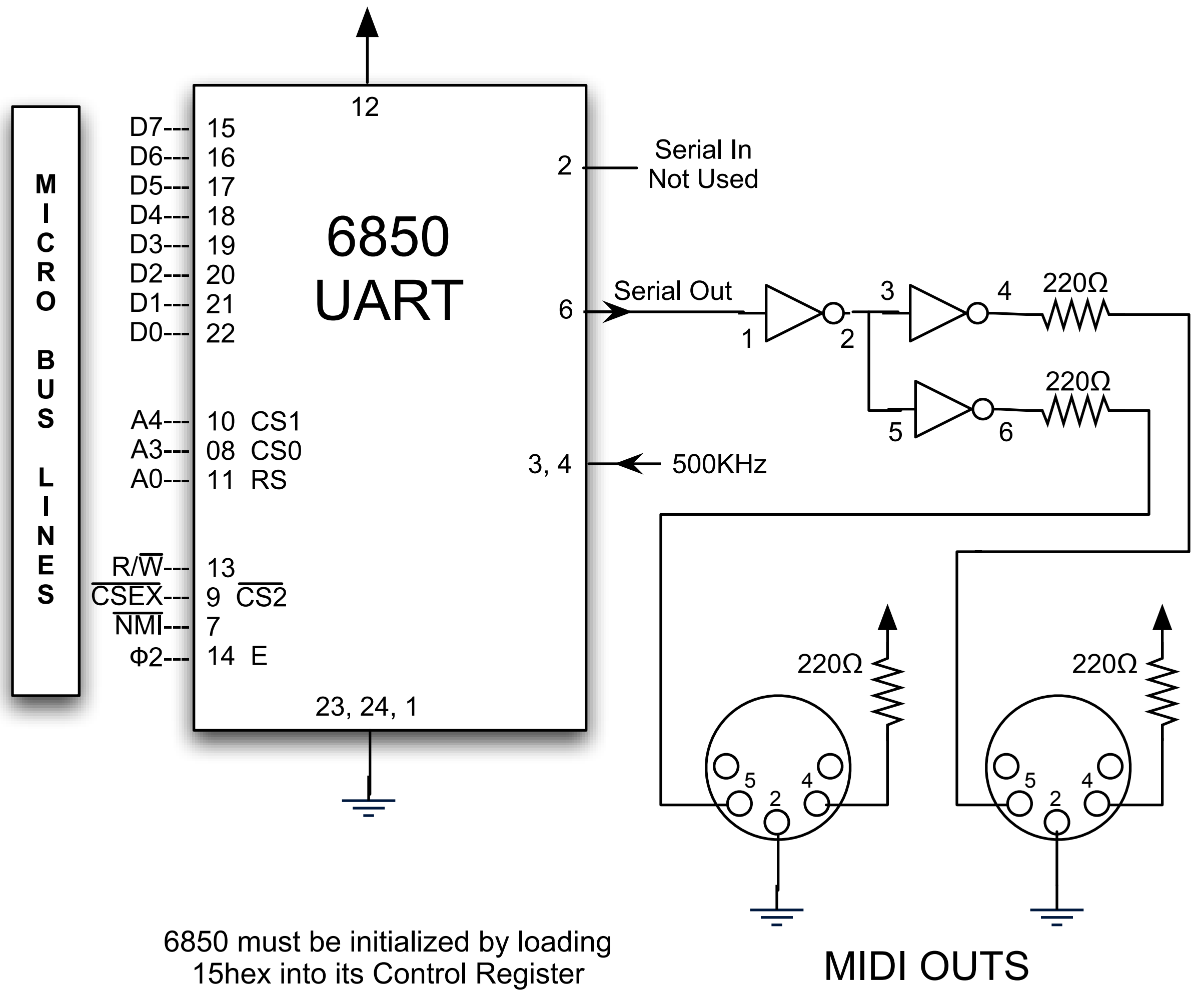
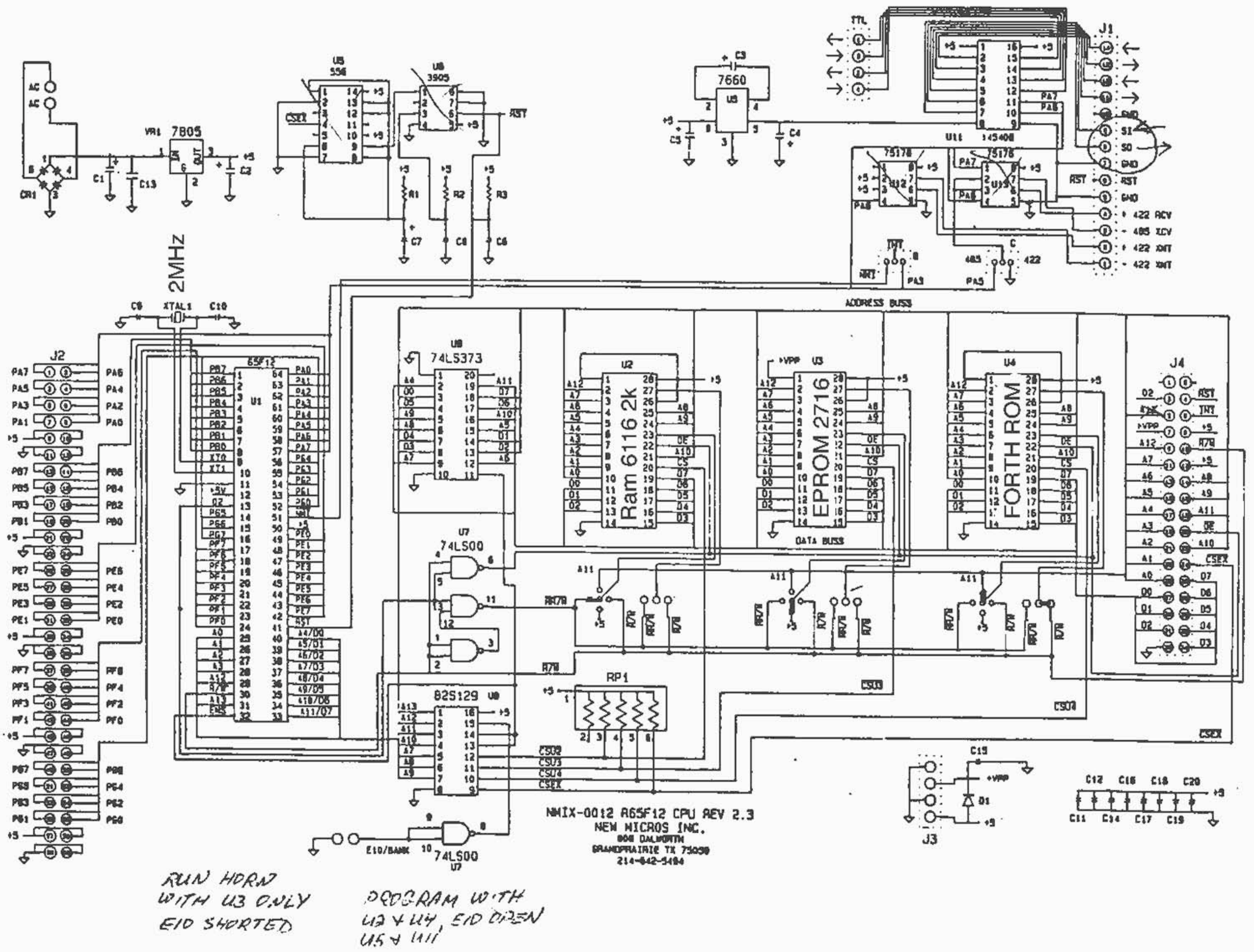


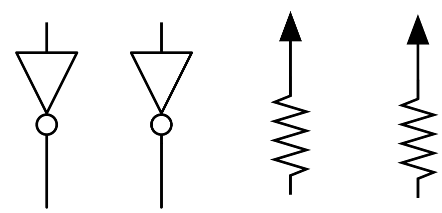
FIGURE 9 - MIDI Interface



2MHz
XTAL1
7805
556
3905
7660
75176
14540B
J1
J2
J3
J4
TTL
65F12
74LS373
Ram 6116 2k
EPROM 2716
FORTH ROM
74LS00
82S129
RP1
ADDRESS BUSS
DATA BUSS
NMIX-0012 R65F12 CPU REV 2.3
NEW MICROS INC.
RUN HORN
WITH U3 ONLY
EID SHORTED
PROGRAM WITH
U3 + U4, EID OPEN
U5 + U11

Micro R65F12 MIDI HORN
Header Connections
(pin side)

GND	D3
D2	D4
D1	D5
D0	D6
A0	D7
A1	CSEX*
A2	A10
A3	0E*
A4	A11
A5	A9
A6	A8
A7	A13
A12	R/W
-	+5
-	INT
Φ2	RST
NC	NC



- B1/sus1 ---------- MIDI OUTS-------------

1	3	5	7	9	11
2	4	6	8	10	12

- B0/sus0 ped2 ped1 5v GND

GND	5V	G0	G2	G4	G6	GND	5V	F0	F2	F4	F6	GND		
GND	5V	G1	G3	G5	G7	GND	5V	F1	F3	F5	F7	GND		

E6	GND	5V	B0	B2	B4	B6	GND	5V	A0	A2	A4	A6	
E7	GND	5V	B1	B3	B5	B7	GND	5V	A1	A3	A5	A7	

MIDI HORN
The Software

MIDI HORN PROGRAM
(in Forth Programming Language)

```
HEX
354 CONSTANT    ?ON        (note on = 1, note off = 0)
355 CONSTANT    PRG        (current program number)
356 CONSTANT    JKY        (last read f keys, front keys)
357 CONSTANT    GKY        (last read g keys, back keys)
358 CONSTANT    BKY        (last read b keys, pedal switches)
359 CONSTANT    FKEY       (f keys currently in effect)
35A CONSTANT    GKEY       (g keys currently in effect)
35B CONSTANT    BKEY       (b keys currently in effect)
35C CONSTANT    NOTE       (calculated key on value)
35D CONSTANT    DB         (breath difference)
35E CONSTANT    VEL        (key on key velocity)
35F CONSTANT    CHNL       (midi channel number)
370 CONSTANT    ?CTL       (true if a midi control status
                            has been sent)
100 CONSTANT    N          (base address of ADC's)
4   CONSTANT    ON_TH      (on threshold at 8C8)
3   CONSTANT    OFF_TH     (off threshold at 8D7)

CV 360 + ;      (control voltages 0-7, at 360-367)
CCV 368 + ;     (current control voltages, at 368-36F)

DECIMAL
: TABLE <BUILDS 0 DO C, LOOP DOES> + C@ ;

1 6 4 9 2 7 5 10 3 8 6 11 4 9 7 12  16 TABLE NT  (note table)
24 60 36 72 48 84 12 96              8 TABLE OCT (octave table)
19 18 4 1 0 17 16 2                  8 TABLE CTL (control #s)

HEX

: CWAIT    (---| wait for ADC conversion to finish)
     BEGIN PA C@ 2 AND UNTIL ;

: CONV     (n ---| start converting ADC number n)
     N + 0 SWAP C! ;

:STORE     (n ---| store ADC number n into CV variables)
     CV N C@ 1 RSHIFT SWAP C! ;  (toss LSB for 7 bit MIDI value)
```

```
CODE  LSHIFT    (x,n --- x'| shift x left by n bits)
     TOP LDY, BEGIN, DEY, 0< NOT WHILE, CLC, SEC ROL,
     SEC 1+ ROL, REPEAT, POP JMP,
END-CODE

CODE  RSHIFT    (x,n --- x'| shift x right by n bits)
     TOP LDY, BEGIN, DEY, 0< NOT WHILE,
     SEC 1+ LSR, SEC ROR, REPEAT, POP JMP,
END-CODE

: MIDINT   (---| initialize midi and data ports)
     15 118 3 OVER C! C!  FF PB C!  FF PF C!  FF PG C!
     0 ?ON C!  0 ?CTL C! ;

: MLD      (x ---| wait for transmit clear then send midi x)
     BEGIN 118 C@ 2 AND UNTIL 119 C! ;

: M        (x1,x2,stat---| send midi staus and 2 data bytes)
     CHNL C@ OR MLD  MLD  MLD ;

: ON       90 M ;           (vel,key---| send midi key-on)
: OFF      0 SWAP 80 M ;    (key---| send midi key-off)
: PROG     CHNL C@ C0 OR MLD MLD ;     (p---| send midi
                                        program change)

: CONT     (val,ctl,---| send midi control change)
     ?CTL C@    (send status byte only once)
     IF ELSE   CHNL C@ B0 OR MLD   THEN
     MLD  1 ?CTL C!   MLD ;

: FORGET   (redefine to allow forgetting colon definitions
            entered with mistakes)
     334 @ DUP C@ DF AND SWAP C! FORGET ;

: GLD      (---| check for changes in back keys and send
            midi control)
     GKY C@ GKEY C@ XOR DUP  IF
     8 0 DO  DUP (xor) 1 I LSHIFT AND
          IF 1 I LSHIFT  GKY C@ AND
               IF  0 54 I + CONT
               ELSE  7F 54 I + CONT  THEN
          THEN
     LOOP  GKY C@ GKEY C!  THEN DROP (xor) ;
```

```
: BLD      (---| check for changes in pedal keys and
                send midi control)
    BKY C@ BKEY C@ XOR DUP  IF
    2 0 DO  DUP (xor) 1 I LSHIFT AND
         IF 1 I LSHIFT  BKY C@ AND
              IF  7F 40 I + CONT
              ELSE  0 40 I + CONT  THEN
         THEN
    LOOP  BKY C@ BKEY C!  THEN DROP (xor) ;

: SEND     (n---| if it has changed send ADC #n's value
           out midi control)
    DUP  CV C@ OVER  CCV C@ OVER  - ABS 2 <
    IF DROP DROP   (don't do anything if change < 2)
    ELSE  SWAP 2DUP  CCV C!   (store it in CCV)
         CTL CONT    (send midi control change)
    THEN ;

: READ     (---| read all horn parameters)
    0 ?CTL C!
    PF C@ FKY C2 = (leave true flag if front keys are new)
    0 CONV
    PG C@ GKY C!  GLD
    PB C@ BKY C!  BLD
    8 0 DO  I STORE  0 I 1+ N + C! (conv)  I SEND  LOOP
    IF ELSE PF C@ FKY C! THEN ; (let key bounce settle
                                 before loading value)

: PROGRAM   (---| read bottom front key for program change)
    FKY C@  80 AND
    IF ELSE FKY C@ PRG C@ = (note keys are read inverted)
         IF ELSE FKY C@  DUP 7F XOR PROG  PRG C! THEN
    THEN ;

: KEYCALC       (---| calculate key-on note value)
    FKY C@  FF XOR DUP  78 AND  3 RSHIFT NT
    SWAP 7 AND OCT  + NOTE C! ;

: BREATH  0 CV C@ ;

: SETVEL   (---| calculate key-on velocity from breath)
    0 CONV BREATH N C@ 1 RSHIFT  MAX 10 + 7F MIN VEL C! ;
```

```
: KEYON    (---| send mid key-on, store key value
             and ?ON flag)
    VEL C@ NOTE C@ ON  1 ?ON C!  FKY C@ FKEY C! ;

: KEYOFF   (---| send midi key-off, store ?ON flag)
     NOTE C@ OFF  0 ?ON C! ;

: SETUP    (---| set midi chanel at startup)
    PF C@  FF XOR 3 RSHIFT F AND CHNL C! ;

: HORN     (---| MAIN PROGRAM)
    SETUP MIDINT BEGIN
    READ PROGRAM BREATH  ?ON C@
    IF (breath) OFF_TH >
          IF FKEY C@ FKY C@ =
               IF ELSE  KEYOFF KEYCALC  BREATH VEL C!
                          KEYON THEN
          ELSE KEYOFF
          THEN
    ELSE (breath) ON_TH >
          IF KEYCALC SETVEL KEYON THEN
    THEN  ?TERMINAL UNTIL ;
```

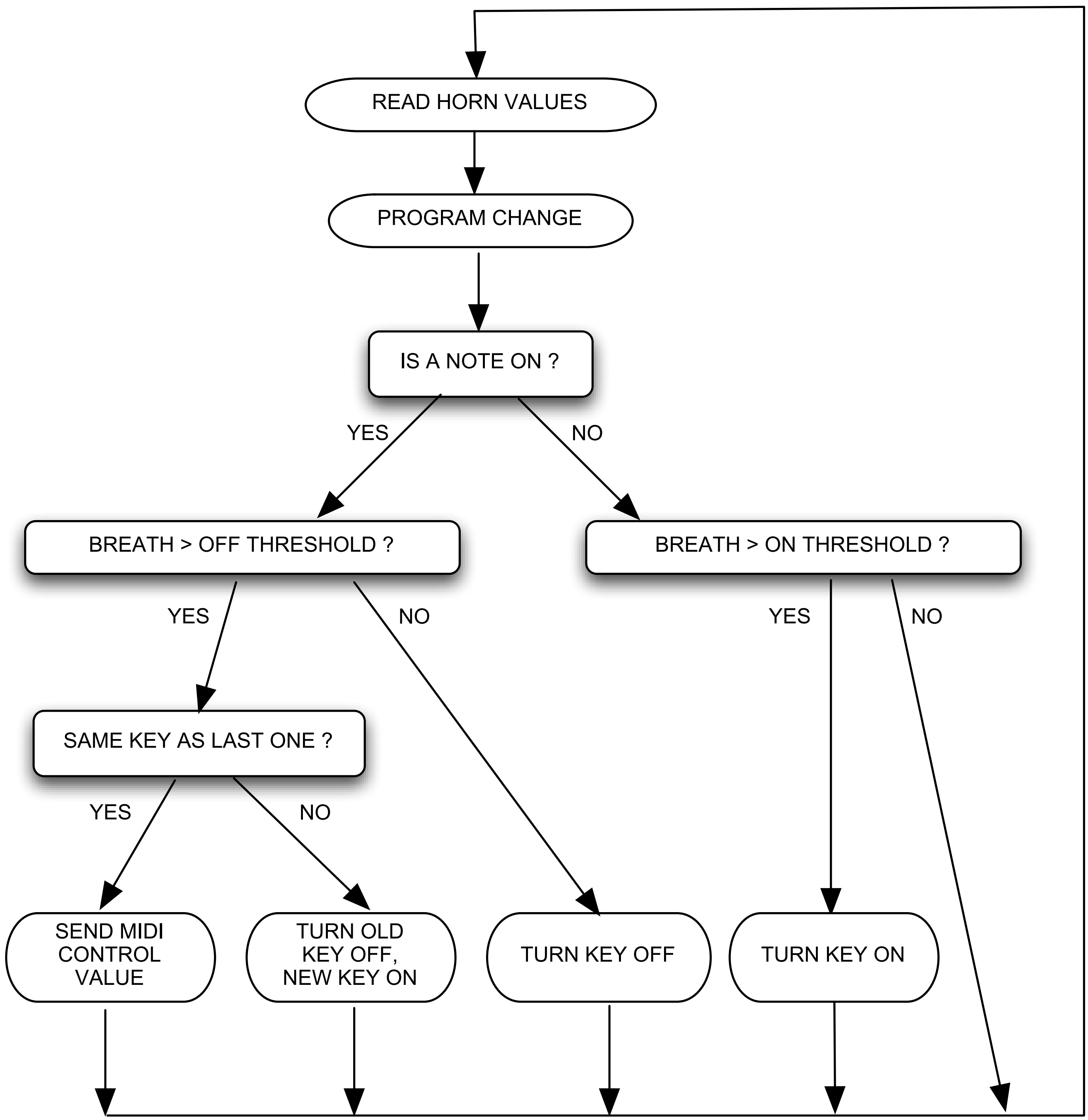
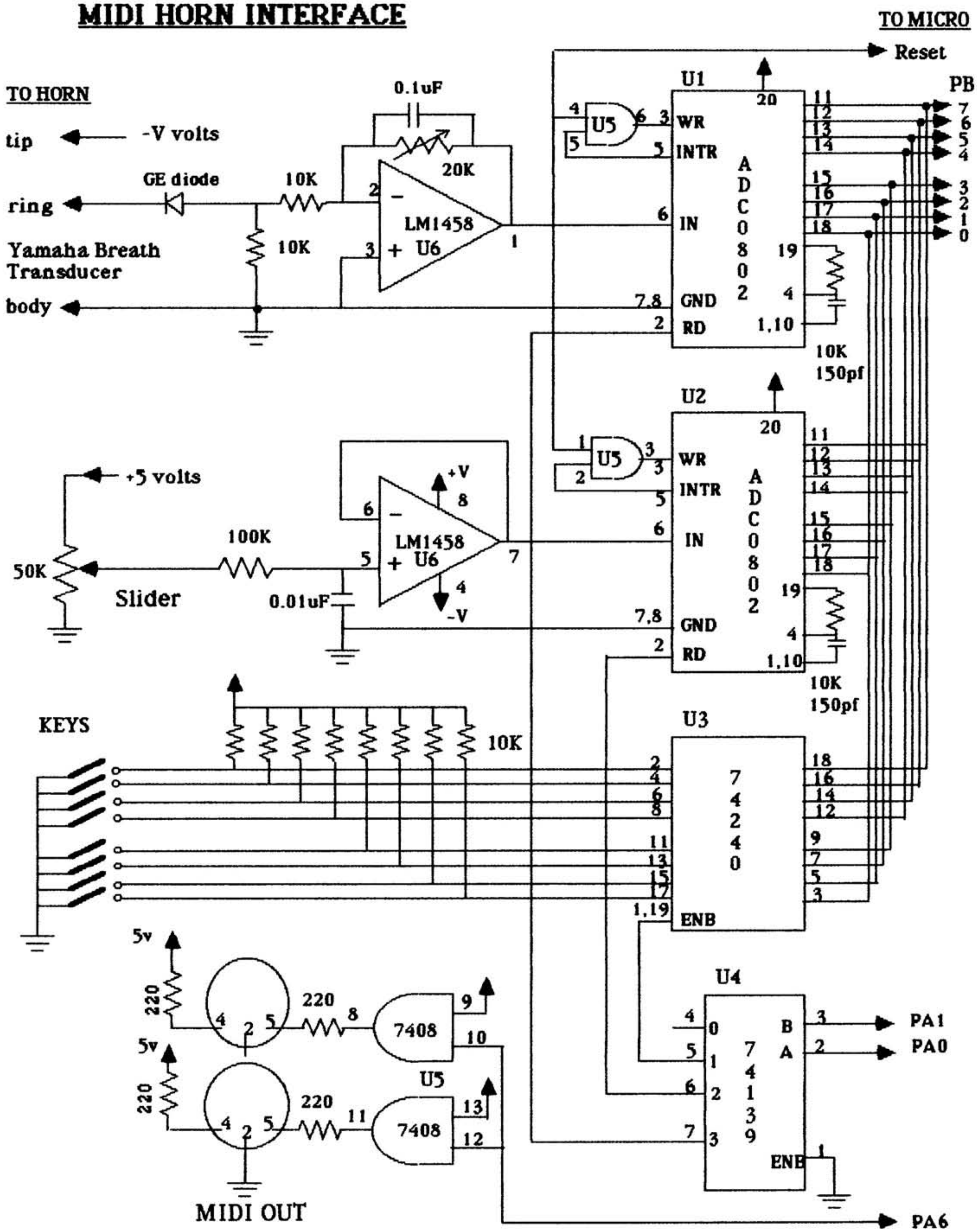


FIGURE 4 - Program Flow Diagram

OLDER MIDI HORN

Simpler version built with a Yamha Breath Transducer, one slide pot on the back, and eight keys on the front

MIDI HORN INTERFACE



```
(This is the program for a smaller version of the MIDI Horn built
 with a Yamaha Breath transducer, one slide pot and 8 keys)

HEX
354 CONSTANT ?ON     ( Is there a note on?  no=0, yes=1 )
355 CONSTANT PRG     ( The current Midi program number )
356 CONSTANT DXKEY   ( The currently on Horn key value )
357 CONSTANT XXKEY   ( The Midi key value calculated from DXKEY )
358 CONSTANT KYBD    ( The last read Horn Key value )
359 CONSTANT SLIDER  ( The last read Horn Slider value )
360 CONSTANT BREATH ( The last read Horn Breath value )

: RD PA C! PB C@ ;  ( Sets up the ports for reading Horn values )

: READ              ( Reads the Horn values and stores them )
    FF RD BREATH C!  FE RD SLIDER C!   FD RD DXKEY C@ =
    IF ELSE 18F 0 DO LOOP     ( Delay for cancelling between key blips )
    FD RD KYBD C! THEN ;

: TERM   C4 SCCR C!   33 18 C!  ;   ( Return serial to terminal on exit )

: MIDINIT   FF PB C!  0 ?ON C!  C0 SCCR C!  1 18 !  ;  ( Set up Midi port )

: MLD     ( n --- | Wait till serial chip is ready, then send Midi data n )
    BEGIN  SCSR C@ 40 AND   UNTIL  17 C! ;


DECIMAL

9 CONSTANT  ON-THRESH   ( Thresholds for determining Key On and )

6 CONSTANT OFF-THRESH   ( Key Off from the Horn Breath value)

: TABLE  <BUILDS 0 DO C, LOOP DOES> + C@ ;

( Horn key fingering tables )
1 6 4 9 2 7 5 10 3 8 6 11 4 9 7 12 16  TABLE NOTE     ( top 4 keys )
24 60 36 72 48 84 12 96 8             TABLE OCTAVE ( bottom 3 keys )

HEX

: M  ( d1,d2,chnl, stat --- | Send 1 midi status and 2 midi data bytes )
    SWAP F AND OR MLD 7F AND MLD 7F AND MLD ;

: MM    ( d1, chnl, stat --- | Send 1 midi status and 1 midi data bytes )
    SWAP F AND OR MLD 7F AND MLD ;

: ON      90 M ;     ( vel, key, chnl --- | Send Midi Note On )
: OFF     80 M ;     ( vel, key, chnl --- | Send Midi Note Off )
: KPRES A0 M ;       ( vel, key, chnl --- | Send Midi Poly Key Press )
: CONT   B0 M ;      ( val, ctl,   chnl --- | Send Midi Control )
: PWHL  E0 M ;       ( msb, lsb, chnl --- | Send Midi Pitchwheel )
: PROG   C0 MM ;     ( prog,       chnl --- | Send Midi Program Change )
: CPRES  DO MM ;     ( val,          chnl --- | Send Midi Channel Pressure )
```

```
: KEYCALC          ( Calculate Midi key value from tables and KYBD )
    KYBD C@ DUP    78 AND     S->D 8 U/         NOTE
    SWAP DROP SWAP    7 AND     OCTAVE +     XXKEY C!     ;

: PROGRAM          ( If back horn key is down, change midi prog )
    KYBD C@ 80 AND     IF KYBD C@ PRG C@ =     IF   ELSE
    KYBD C@ DUP     0 PROG    PRG C!    THEN  THEN  ;

: KEYON            ( Send midi key on, key vel from slider )
    SLIDER C@ XXKEY C@ 0 ON   KYBD C@ DXKEY C!   1 ?ON C!  ;

: KEYOFF           ( Send midi key off )
    0 XXKEY C@ 0 OFF    0 ?ON C!   ;

: CONTROL          ( Send midi control change, Breath control used )
    BREATH C@ 2 0 CONT  ;

: HORN             ( Main Program Loop, exit by keying terminal )
    MIDINIT   BEGIN   READ   PROGRAM   ?ON C@
    IF    BREATH C@ OFF-THRESH >
           IF   DXKEY C@ KYBD C@ =
                 IF CONTROL ELSE KEYOFF KEYCALC KEYON THEN
              ELSE  KEYOFF
           THEN
    ELSE    BREATH C@ ON-THRESH >     IF KEYCALC KEYON THEN
    THEN   ?TERMINAL   UNTIL   TERM   ;


( Two Test Programs )

: SCALE  MIDINIT  BEGIN   60 10 DO   40 I 0 ON  4FF 0 DO LOOP
    0 I 0 OFF  ?TERMINAL  IF LEAVE THEN  LOOP ?TERMINAL UNTIL TERM ;`
: HORN.   CR BEGIN READ SLIDER C@ .  BREATH C@ .  KYBD C@ .
    D EMIT  ?TERMINAL UNTIL ;
```