

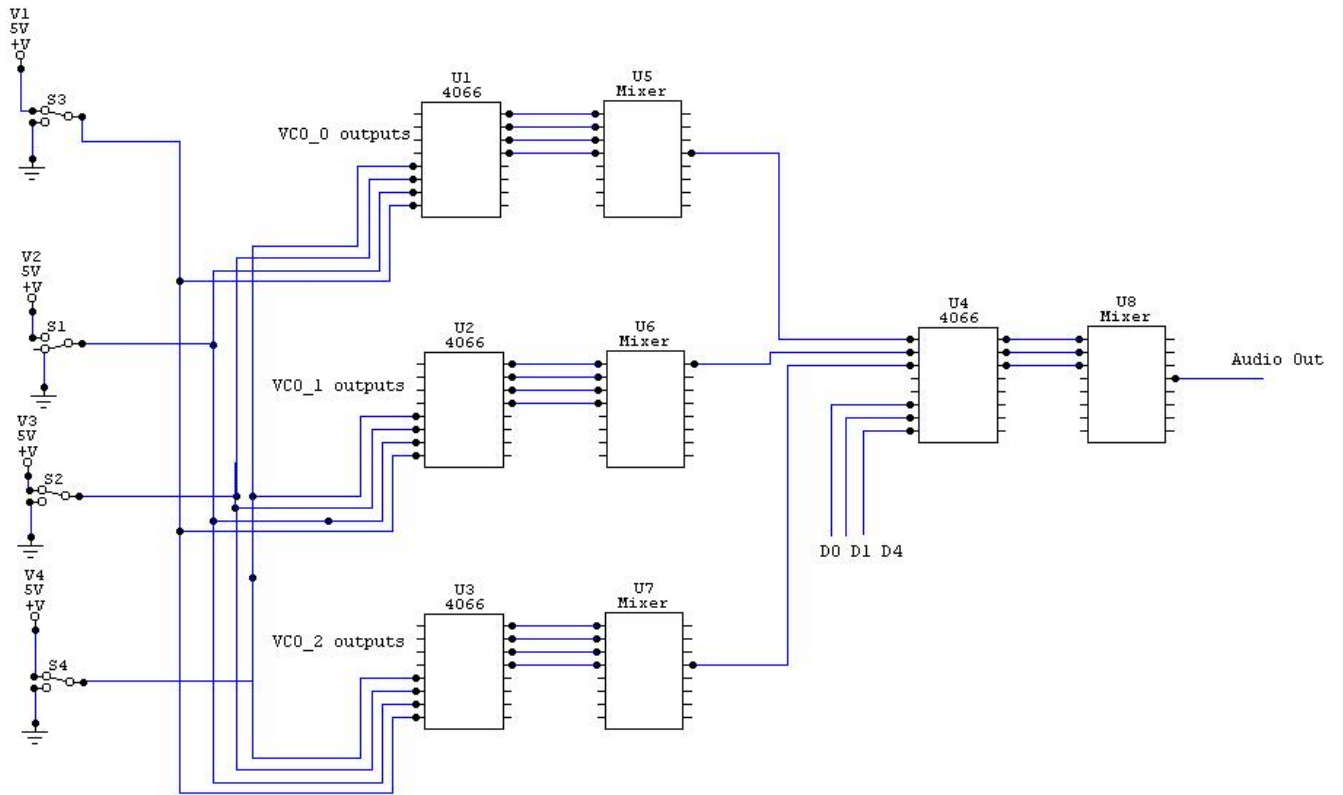
## The Future of Sound...NOW!

Andy "Machine" Gunn – System Integration  
"Chief" Peter Drembelas – Hardware  
Erik "Tex" Czernikowski – Hardware  
Jeremiah "Nullsleep" Johnson – Software  
Olga Z"80"aitseva – Software

The first part of the project was devoted to building the core: the Z80 output connected to an 8-bit D/A converter, which created a DC voltage level for the Voltage-Controlled Oscillator (VCO), whose output ran through a simple Voltage-Controlled Filter (VCF) implementation and through to an audio device. The software to control the output stage and a simple software sequencer was created at this point to test the initial hardware. After this stage was functional, more features were added, such as mixer and voice-gating circuitry, and a MIDI-control section. The hardware and software for the MIDI was the most difficult section to implement. In the end, the MIDI features worked per our specifications. We achieved a three-voice polyphonic analog synthesizer with working MIDI control capabilities and simultaneous software-based sequencing. We had front-panel controls for the Voltage-Controlled Filter (both for resonance and for control voltage), for the pulse-width of the VCOs, and several switches to control the waveform selection as well as the synchronization of the VCOs.

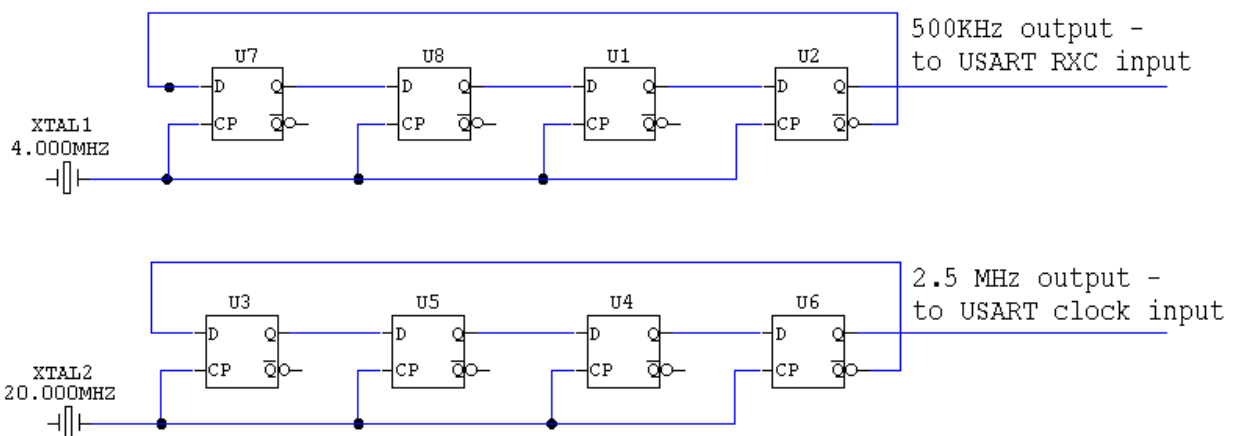
## **Analog Switches for Output Stage:**

The following is the logic used to control the analog switches for the output stage of the synthesizer. We used four DPDT switches tied to high and low voltages to select the waveform type for the set of three VCOs (square wave, sine, triangle, or ramp) via three CD4066 analog switches. The outputs of the 4066s were sent into a mixer (one for each 4066) as a way to multiplex the four signals into one output. The three mixer outputs were then in turn sent to another 4066 as a way to eliminate the ambient hum emitted by each VCO when not receiving control voltages; via  $D_0$ ,  $D_1$ , and  $D_4$ , the Z80 could use the 4066 to select only those VCO outputs that were used for playing a note at the current time.



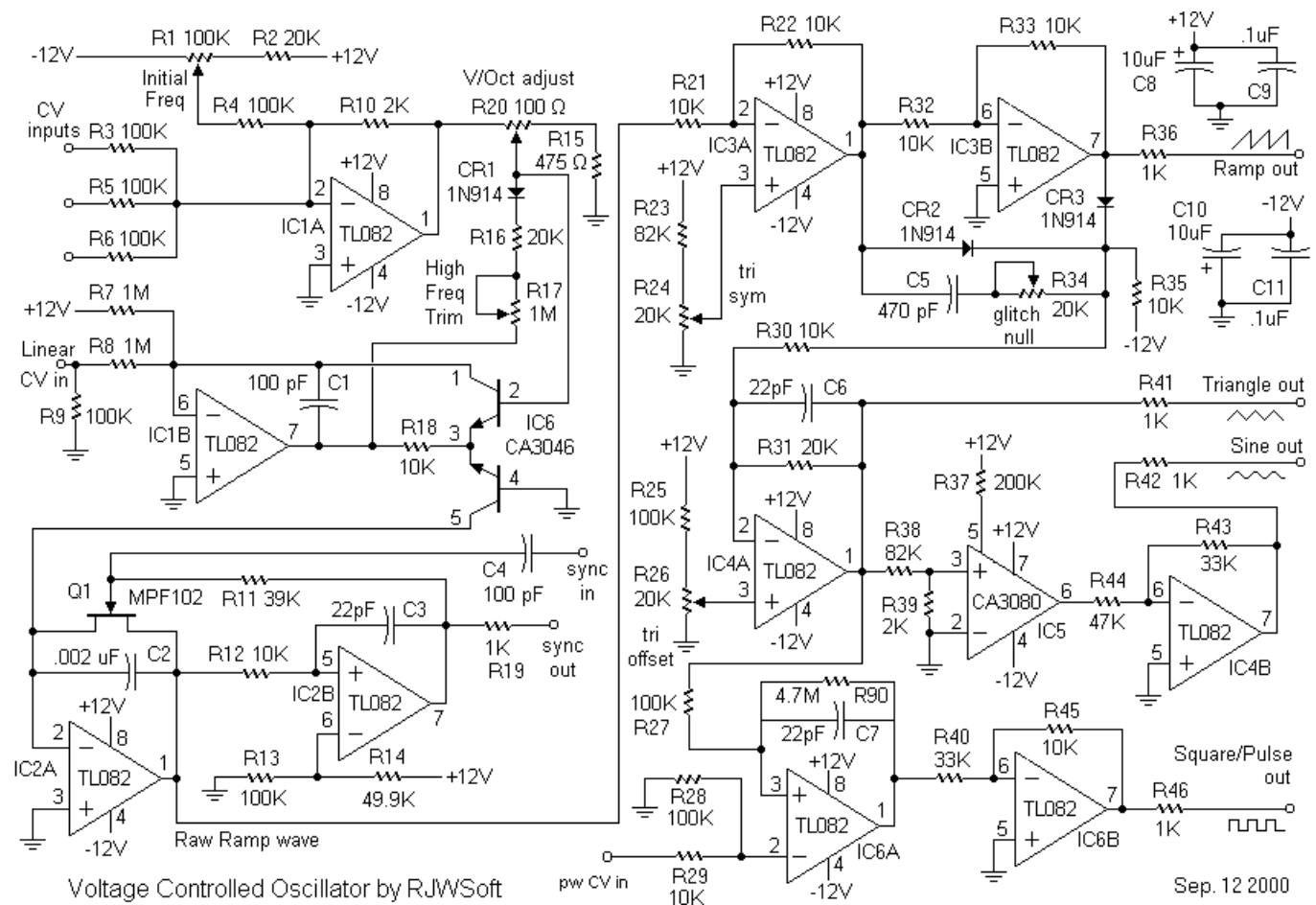
## Quartz crystals and dividing circuitry for USART clock signals:

The 8251 USART chip we employed in our MIDI input circuit required two clock signals: one receive clock at 500 kHz to be used to sample MIDI data 16 times per bit (MIDI standard uses a 31.25 kHz frequency to send data), and the other as a master clock for the USART itself, which was to be at least five times the receive clock rate (i.e, 2.5 MHz), but no more than 3.125 MHz. We therefore made use of two crystals, a 4 MHz and a 20 MHz, both stepped down by a factor of 8 to yield frequencies of 500 kHz and 2.5 MHz, respectively. Due to the precision of the 16x sampling rate, synchronization of the two clock signals was deemed unnecessary.



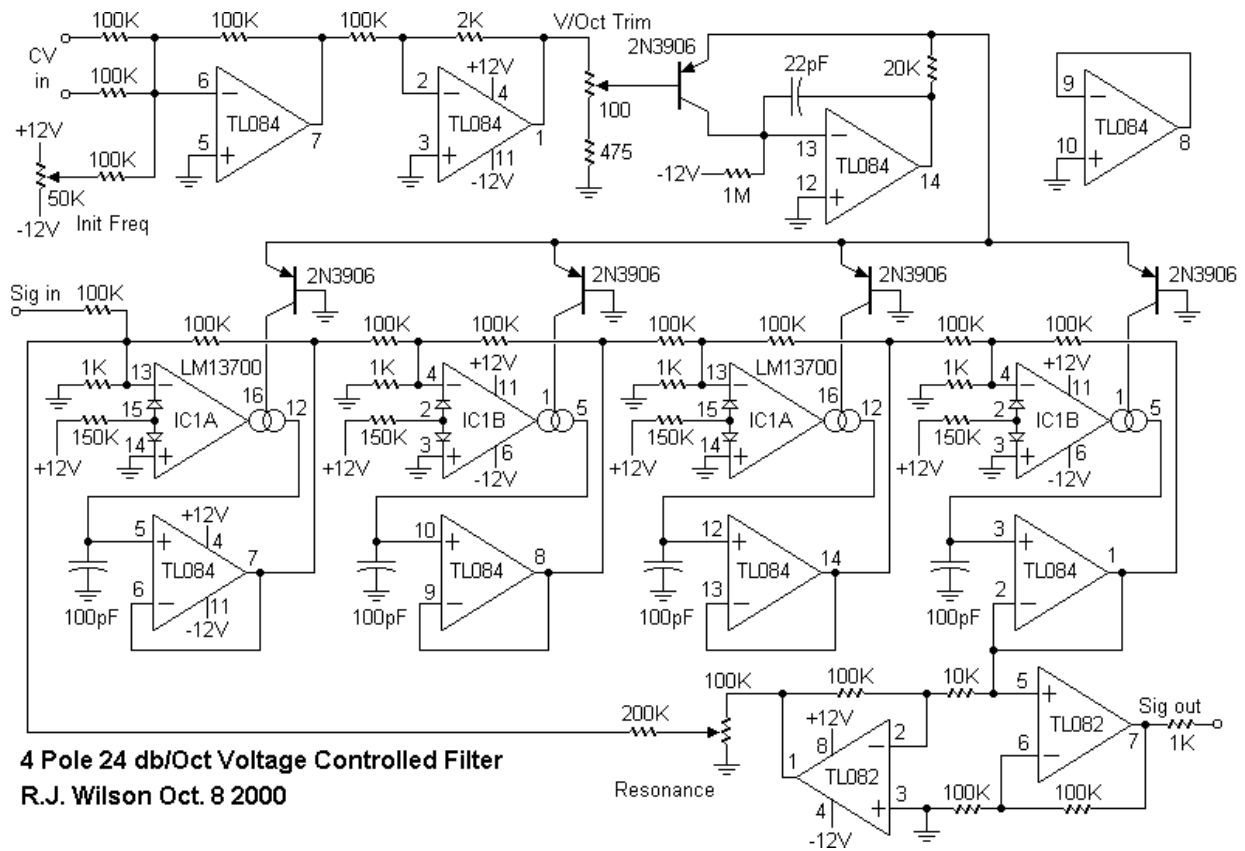
## Voltage-Controlled Oscillator circuit:

In order to generate tones, we made use of three Voltage-Controlled Oscillator circuits (VCOs), each of which produced four different waveform outputs, including a sine wave, a square wave, a triangle wave, and a ramp wave. The VCOs operated using a control voltage as input, which was applied via a digital-to-analog converter (DAC) and a scaling amplifier that normalized the analog output to between 0 and 5V for the 8-bit input range it received. The potentiometers for base frequency and for voltage-per-octave (see below) were adjusted accordingly to achieve a four-octave spread over the 5V control voltage range. The VCOs also had capabilities for synchronization with other VCOs via the “sync in” and “sync out” ports: one VCO would be designated as “master” and the other as “slave”, whereby the master VCO would send a pulse out to the sync-in port of the slave oscillator(s) each time it completed a period of its waveform. This pulse would force the slave oscillator to restart the period of its own waveform, which would add harmonics to the signal and yield a “space-music”-esque sound. nb: the 22 $\mu$ F capacitor in the lower left labeled “C3” was found to be too low to yield a time constant that would result in audible output frequencies; as such, we added another 22 $\mu$ F capacitor in parallel.



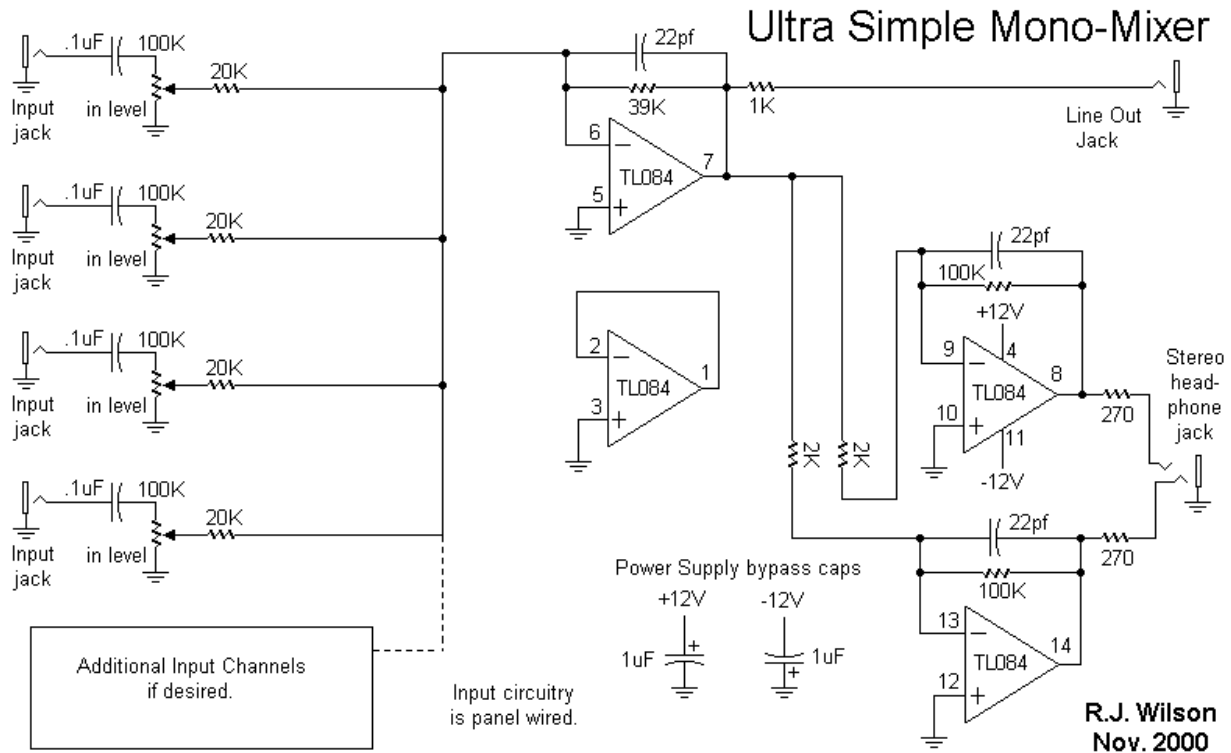
## Voltage-Controlled Filter circuit:

The voltage-controlled filter (VCF) was used in the output stage to offer the possibility for different signal characteristics and thus different sound qualities for the synthesizer. It operates with a control voltage input as well as an input waveform signal, and it makes use of two potentiometers to adjust both the initial frequency and the resonance of the output signal.



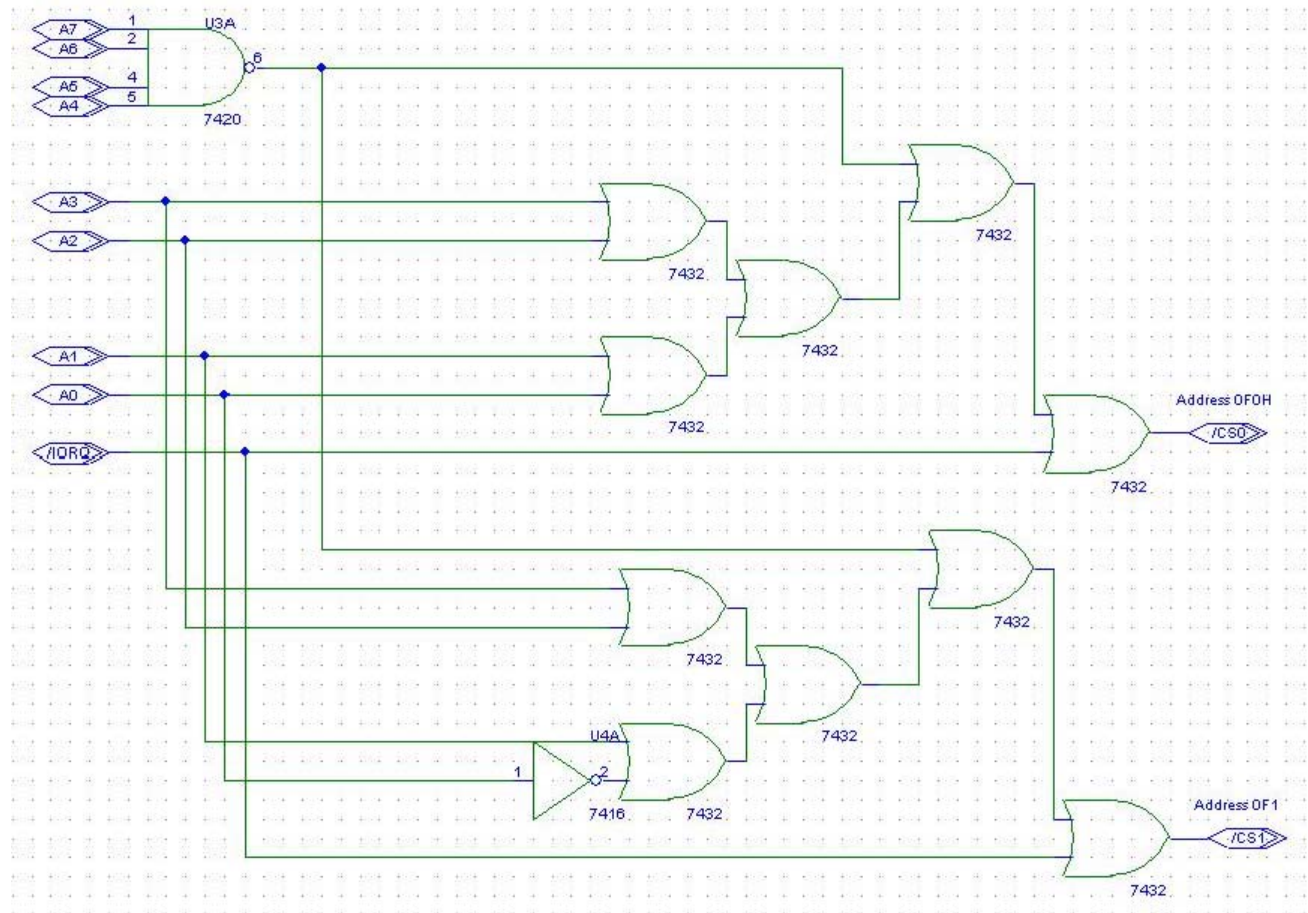
## Mono Mixer circuit:

We employed several of the mixer circuits to mix several audio outputs into one output signal. In the intermediate stage, the mixers were used as multiplexers to channel the four outputs of the VCOs into one, such that it would make selection of the waveform easier to implement. The other mixer was used in the final stage to allow the three voices to be sent through to one audio output and thus one set of speakers. As we did not make use of stereo signals, we did not implement the stereo headphone jack part of the mixer circuit.



## Port-Decoding Logic:

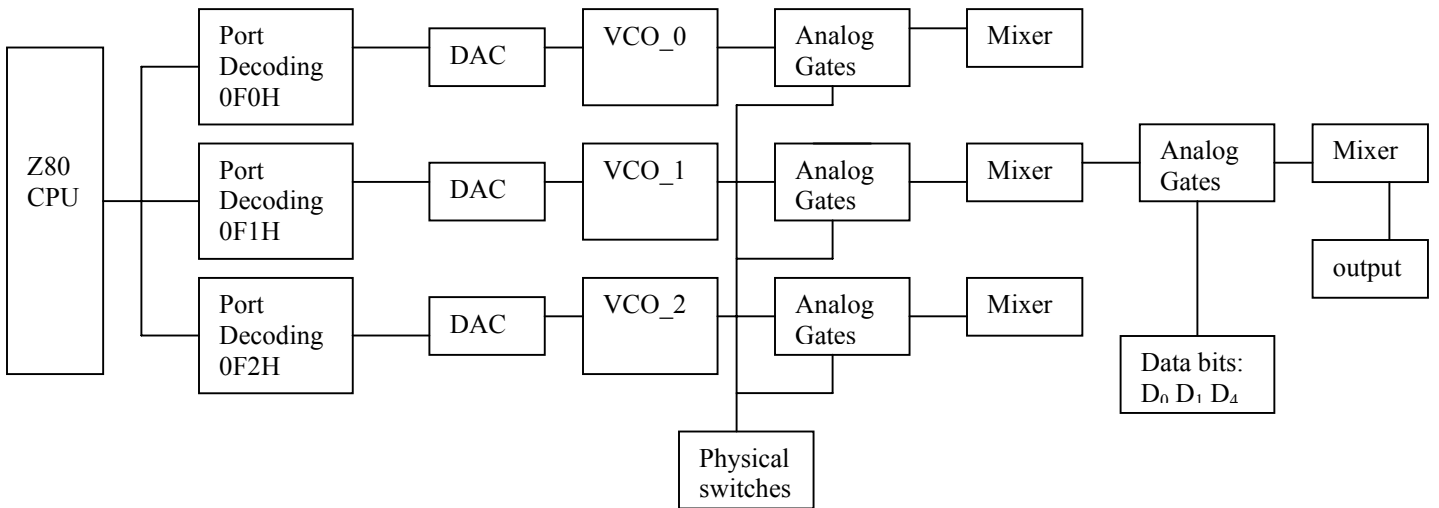
In order to control the various circuit components of the synthesizer, we used memory-mapped port addressing via decoding logic like the circuit below. For these, we simply used standard TTL logic ICs and employed active low chip-select signals for the DACs and the USART. The port addresses used are listed below as well.



A7	A6	A5	A4	A3	A2	A1	A0	Device	Port Num
1	1	1	1	0	0	0	0	DAC chip select for VCO <sub>0</sub>	F0H
1	1	1	1	0	0	0	1	DAC chip select for VCO <sub>1</sub>	F1H
1	1	1	1	0	0	1	0	DAC chip select for VCO <sub>2</sub>	F2H
1	1	1	1	0	1	0	0	Chip select for Analog Switches	F4H
1	1	1	1	1	0	0	0	USART Data Register Port	F8H
1	1	1	1	1	0	0	1	USART Control Register Port	F9H

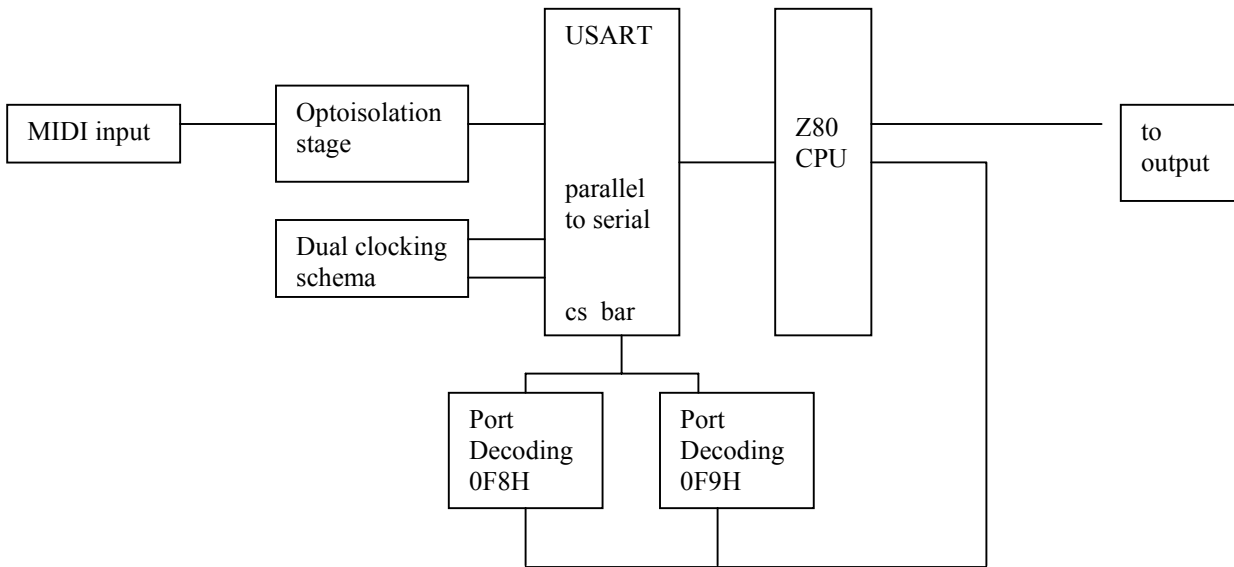
## Zynth80 Output Stage:

The following is a high-level diagram for the output stage of the Zynth80. The Z80 processor is interfaced with the port decoding logic sections which control the Digital-to-Analog converters. An 8-bit word is sent to the inputs of the DACs and converted to an analog voltage, which is amplified to a 0-5V scale. This analog voltage acts as the control voltage for a VCO, whose outputs are gated via the circuit above (waveform select via the physical switches, voice select via the data bus).



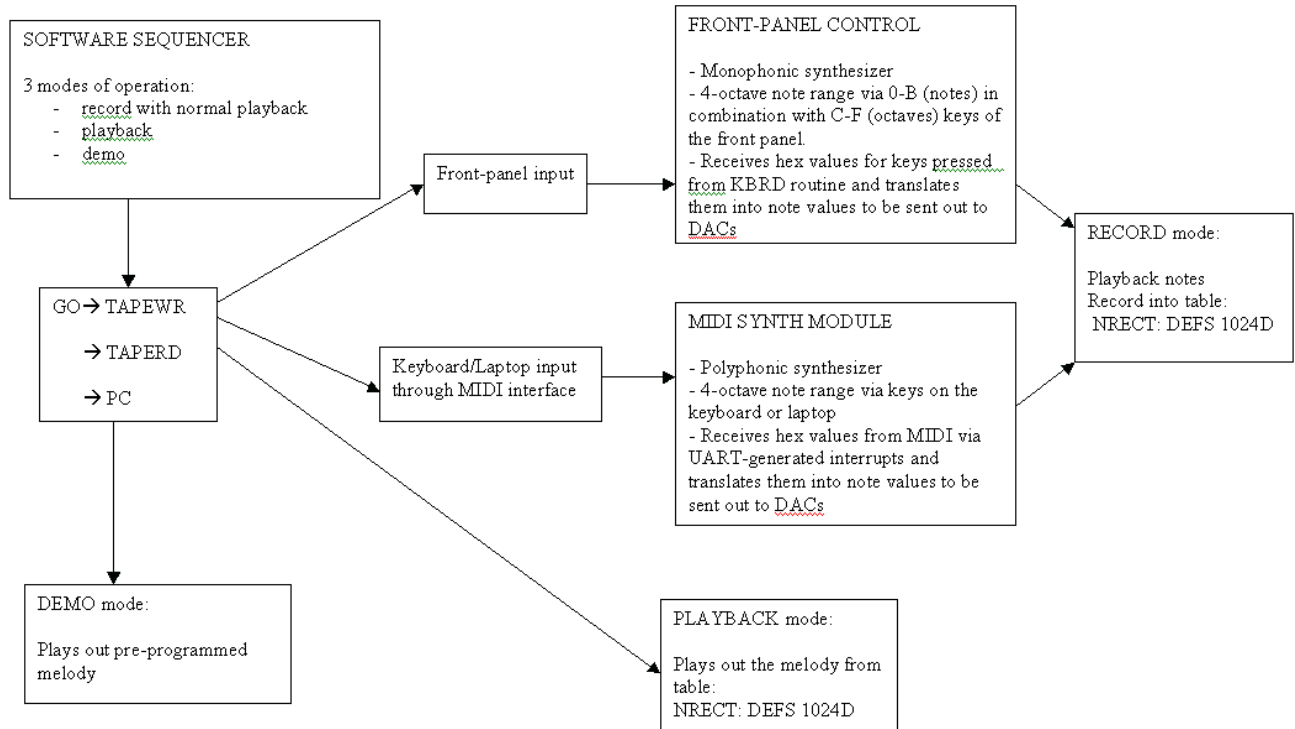
## Zynth80 Input Stage:

The following is a high-level diagram for the input stage of the Zynth80. MIDI input is received at 31.25 kHz from a source (a keyboard, a computer) and buffered through a high-speed Darlington photocoupler, which generates the necessary digital voltages and is sent into the USART chip (which is clocked via the crystal circuitry described above). The USART is controlled by two sets of port-decoding logic (one for the control signals, one for the data), and its output is sent via the data bus to the Z80 kit, which processes it and send the necessary signals along to the DACs for conversion into musical signals.



## Software Flow-Chart:

The following is a flow-chart representing the software component of the Zynth80. The software sequencer was implemented through front-panel keypad control and keyboard/any other MIDI interface. During the first stage of the project front-panel control sequencer was used to test and tune the hardware components as well as the range of possibilities in the implementation. During the MIDI integration stage code was translated as closely as possible to work with MIDI signals while retaining all original features.



## Front-Panel Control

### Implementation:

The initial direction of software development was aimed at creating a program capable of allowing the user to play notes through input from the front panel of the microtrainer. The program would also be capable of recording the key presses and saving them for subsequent playback. Additionally, a final mode would allow the user to playback a pre-recorded/programmed song that is held within program's memory. All of these goals for the zynth80 were fully realized, only after some debugging related to hardware integration. The front panel allows all notes within a range of 4 octaves to be played back by the user with keymapping was as follows:

Key Press	Key Function
0	C
1	C#
2	D
3	D#
4	E
5	F
6	F#
7	G
8	G#
9	A
A	A#
B	B
C	Octave 1
D	Octave 2
E	Octave 3
F	Octave 4

The implementation was accomplished relatively early along in the course of the project by running a lookup on a series of tables containing voltages to be sent to the DACs for all four octaves. When the KBRD call is issued, and the user presses one of the valid key for note playback the hex value sent by the keypress is used as an index offset for the table lookup. Additionally, the keypress data is saved into another table NRECT that has a maximum size of 1024 bytes. During playback selection each value stored in this table is sequentially and used as an offset for looking up corresponding voltages to be sent to the DACs, just as if the user were entering the keypresses in real-time. The pre-programmed demo song operated similarly. This full-featured program using the front panel as an input source was very useful once the MIDI implementation began.

## Assembly code:

```
;;Software sequencer implementation with input through front-panel  
;;keys of the Z80.
```

```
DELAY      EQU 0FFDH  
PORT1      EQU 0F0H  
KBRD       EQU 0FEBH  
SCAN       EQU 0FEEH  
MESOUT     EQU 0FF4H  
TONE       EQU 0FFAH  
HEXTO7     EQU 0FF1H  
TWRKEY     EQU 1EH  
TRDKEY     EQU 1FH  
GOKEY      EQU 12H  
FKEY       EQU 0FH
```

```
ORG 1800H  
LD SP, 1F00H  
LD A, 80H  
OUT (0C3H), A  
LD HL, 250D
```

```
;;PRESS routine that scans the front-panel for user input expecting  
;;TPWRITE, TPREAD, or F keys to be pressed
```

```
PRESS:
```

```
LD IX, SEQST  
CALL MESOUT  
CALL KBRD      ;WAIT TILL USER PRESSES A KEY  
CP TWRKEY      ;IF IT'S TWRKEY THEN GO TO RECORD  
JP Z, RECORD  
CP TRDKEY      ;ELSE IF IT'S TRDKEY THEN GO TO PLAYBK  
JP Z, PLAYBK  
CP FKEY        ;ELSE IF IT'S FKEY THEN GO TO PLREC  
JP Z, PLREC  
JP PRESS      ;ELSE LOOP BACK
```

```
;;Routine to play out pre-programmed melody
```

```
PLREC:
```

```
PUSH HL  
LD IX, PLRST  
CALL MESOUT  
CALL KBRD      ;CHECK USER INPUT  
CP GOKEY       ;if GO was pressed, return to main menu  
JP Z, EXTPLR  
CP FKEY        ;if F was pressed proceed, o.w. wait for  
JP NZ, PLREC   ;correct input  
LD IY, RECT    ;point IY pointer to RECT table, containing  
               ;pre-recorded melody  
PLR: LD B, (IY+0) ;GET NOTE  
LD A, B  
CP 0FFH       ;CHECK IF IT'S AT THE END OF THE TABLE  
JP Z, PLREC    ;if it is return to PLREC  
INC IY  
LD L, 00D  
LD H, (IY+0)   ;GET DURATION (HIGH BYTE)
```

```

LD A, B
OUT (PORT1), A      ;OUTPUT THE NOTE
OUT (0C0H), A
CALL DELAY          ;FOR A GIVEN DURATION
INC IY
JP PLR
EXTPLR:
POP HL
JP PRESS

;;Routine to record a melody by accepting user key presses
RECORD:
LD IY, NRECT        ;POINT IY TO NRECT TABLE TO RECORD NOTES IN
LD (IY+0), 0FFH     ;0FFH="END OF TABLE"
LD D, 0CH           ;DEFAULT OCTAVE=1
REC: LD IX, RECST
CALL MESOUT
CALL KBRD
CP GOKEY            ;CHECK IF GO KEY IS PRESSED
JP Z, PRESS         ;IF IT IS, DONE RECORDING
CALL OCCHECK        ;O.W. CHECK IF ACCEPTABLE OCTAVE IS PRESSED
JP Z, RESETOC       ;IF SO RESET OCTAVE
CALL NCHECK         ;O.W. VERIFY THE NOTE IS WITHIN RANGE
JP NZ, REC
LD C, A             ;SAVE NOTE INTO BC
LD B, 0
LD A, D             ;FIND OCTAVE
CP 0CH
JP Z, LDIX1
CP 0DH
JP Z, LDIX2
CP 0EH
JP Z, LDIX3
CP 0FH
JP Z, LDIX4
REC2: ADD IX, BC
LD A, (IX+0)        ;load the note into A
LD (IY+0), A        ;PUT NOTE INTO NRECT TABLE
LD (IY+1), 0FFH     ;assign "end of table" to next note
INC IY              ;inc IY TO NEXT POINT IN TABLE
OUT (PORT1), A      ;OUTPUT NOTE
OUT (0C0H), A
CALL DELAY
JP REC              ;continue recording

LDIX1:
LD IX, PTRN1        ;POINT IX TO PTRN1 TABLE TO FIND APPROPRIATE
                        ;NOTE VOLTAGE IN 1ST OCTAVE
JP REC2
LDIX2:
LD IX, PTRN2        ;POINT IX TO PTRN1 TABLE TO FIND APPROPRIATE
                        ;NOTE VOLTAGE IN 2ST OCTAVE
JP REC2
LDIX3:
LD IX, PTRN3        ;POINT IX TO PTRN1 TABLE TO FIND APPROPRIATE

```

```

                                ;NOTE VOLTAGE IN 3ST OCTAVE
JP REC2

LDIX4:
LD IX, PTRN4                ;POINT IX TO PTRN1 TABLE TO FIND APPROPRIATE
                                ;NOTE VOLTAGE IN 4ST OCTAVE
JP REC2

;;Routine to reset octave currently playing
RESETOC:
LD D, A
JP REC

;;Routine to check if correct octave key was pressed
;;keys C-F are accepted as octaves 1-4
OCCHECK:
CP 0CH
RET Z
CP 0DH
RET Z
CP 0EH
RET Z
CP 0FH
RET Z
RET                ;o.w. return with NZ

;;Routine to check if correct note key has been pressed
;;keys 0-B are accepted as notes, everything else causes a tone to sound
NCHECK:
CP 00H
RET Z
CP 01H
RET Z
CP 02H
RET Z
CP 03H
RET Z
CP 04H
RET Z
CP 05H
RET Z
CP 06H
RET Z
CP 07H
RET Z
CP 08H
RET Z
CP 09H
RET Z
CP 0AH
RET Z
CP 0BH
RET Z
CALL TONE
RET                ;return with NZ condition

```

```

;;Routine to play user-entered melody
PLAYBK:
    LD IX, PLAYST
    CALL MESOUT
    CALL KBRD
    CP GOKEY           ;check if GO key is pressed
    JP Z, PRESS       ;if so exit to main menu
    CP TRDKEY         ;if TPREAD pressed, proceed
    JP NZ, PLAYBK     ;o.w. wait for correct input
    LD IY, NRECT      ;POINT TO NRECT TABLE
PLAY: LD A, (IY+0)    ;LOAD NOTE TO PLAY
    CP 0FFH          ;CHECK IF ITS "END OF TABLE"
    JP Z, PLAYBK     ;if so go back to PLAYBK
    OUT (PORT1), A   ;ELSE OUTPUT NOTE
    OUT (0C0H), A
    CALL DELAY
    INC IY           ;IY++
    JP PLAY

```

```

;;RECORDED SONG TABLE
;;NOTE PITCH followed by DURATION HIGH BYTE

```

```

RECT: DEFB 20D
    DEFB 02D
    DEFB 20D
    DEFB 02D
    DEFB 30D
    DEFB 02D
    DEFB 15D
    DEFB 03D
    DEFB 20D
    DEFB 01D
    DEFB 30D
    DEFB 02D
    DEFB 40D
    DEFB 02D
    DEFB 40D
    DEFB 02D
    DEFB 45D
    DEFB 02D
    DEFB 40D
    DEFB 03D
    DEFB 30D
    DEFB 01D
    DEFB 20D
    DEFB 02D
    DEFB 30D
    DEFB 02D
    DEFB 20D
    DEFB 02D
    DEFB 15D
    DEFB 02D
    DEFB 20D
    DEFB 06D
    DEFB 54D
    DEFB 02D

```

DEFB 54D  
DEFB 02D  
DEFB 54D  
DEFB 02D  
DEFB 54D  
DEFB 03D  
DEFB 45D  
DEFB 01D  
DEFB 40D  
DEFB 02D  
DEFB 45D  
DEFB 02D  
DEFB 45D  
DEFB 02D  
DEFB 45D  
DEFB 02D  
DEFB 45D  
DEFB 03D  
DEFB 40D  
DEFB 01D  
DEFB 30D  
DEFB 02D  
DEFB 40D  
DEFB 02D  
DEFB 45D  
DEFB 01D  
DEFB 40D  
DEFB 01D  
DEFB 30D  
DEFB 01D  
DEFB 20D  
DEFB 01D  
DEFB 40D  
DEFB 03D  
DEFB 45D  
DEFB 01D  
DEFB 54D  
DEFB 02D  
DEFB 63D  
DEFB 01D  
DEFB 45D  
DEFB 01D  
DEFB 40D  
DEFB 02D  
DEFB 30D  
DEFB 02D  
DEFB 20D  
DEFB 06D  
DEFB 0FFH ;END OF TABLE MARKER

PLRST: DEFM '-SONG-'

RECST: DEFM 'RECORD'

PLAYST: DEFM '-PLAY-'

SEQST: DEFM 'CHOOSE'

PTRN1:           DEFB 15D       ;KEY=0 NOTE=C  
                  DEFB 20D  
                  DEFB 25D  
                  DEFB 30D  
                  DEFB 35D  
                  DEFB 40D  
                  DEFB 45D  
                  DEFB 49D  
                  DEFB 54D  
                  DEFB 59D  
                  DEFB 63D  
                  DEFB 67D       ;KEY=B NOTE=B

PTRN2:           DEFB 72D  
                  DEFB 77D  
                  DEFB 82D  
                  DEFB 87D  
                  DEFB 92D  
                  DEFB 97D  
                  DEFB 102D  
                  DEFB 106D  
                  DEFB 111D  
                  DEFB 116D  
                  DEFB 120D  
                  DEFB 124D       ;KEY=B NOTE=B

PTRN3:           DEFB 129D  
                  DEFB 134D  
                  DEFB 139D  
                  DEFB 144D  
                  DEFB 149D  
                  DEFB 154D  
                  DEFB 159D  
                  DEFB 163D  
                  DEFB 168D  
                  DEFB 173D  
                  DEFB 177D  
                  DEFB 181D       ;KEY=B NOTE=B

PTRN4:           DEFB 186D  
                  DEFB 191D  
                  DEFB 196D  
                  DEFB 201D  
                  DEFB 206D  
                  DEFB 211D  
                  DEFB 216D  
                  DEFB 220D  
                  DEFB 225D  
                  DEFB 230D  
                  DEFB 234D  
                  DEFB 238D       ;KEY=B NOTE=B

NRECT:           DEFB 1024D   ;location and size of a table to record user-entered  
                                  ;melody

# MIDI Synth Module

## Implementation:

The software for interfacing the MIDI controller with the zynth80 was in many ways similar to the software sequencer, with the higher order of complexity demanded for the integration of the UART and multiple output ports. One of the first things to occur during the execution of the MIDI software is UART initialization. This includes sending three dummy words out in order to accomplish a reset and then issuing the mode and command words for proper setup. The UART would be responsible for receiving the MIDI data from an external controller and issuing interrupts to the z80 containing note on/off, note value, and velocity data values.

The setup of interrupts proved to be more complex than originally anticipated. Initially, we chose to utilize mode 2 interrupts, seeing that they seemed most appropriate and we had already had some experience with them in one of the past lab experiments. However, our first implementation showed a problem that was apparently caused by the inability of the z80 to keep up with the speed at which the UART was issuing interrupts. Subsequently, hardware and software teams attacked the problem simultaneously with two different approaches. While hardware was being built that would use flip-flops to hold onto data longer and provide some extra time for the z80 to take in the interrupts, software utilizing mode 1 interrupts was being coded and tested. Although it was thought that mode 1 interrupts were unusable on the z80 microtrainer kits, we were able to successfully implement a mode 1 interrupt service routine that proved to work excellently for our needs.

Next step was to figure out exactly what signals the MIDI was sending, which was accomplished by first receiving and recording all signals resulting from pressing a sequence of keys and analyzing them. During this step we realized that there were quiet a bit of “extra” signals sent out to initialize all MIDI channels, etc. Fortunately these extra signals were easy to bypass through filtering only for the expected MIDI signals, which consist of 3 8-bit words: status word (note on/off), note pitch value, and note velocity value (we chose to ignore the last one).

Having done the basic integration we proceeded to add onto software to match all functions of the front-panel implementation. Last feature added was software implementation of analog gates to mute/unmute voices, which was accomplished by sending out 8bit control signal to 0F4H port any time there was a change in port status (note on/off).

## Assembly code:

```
;;Software sequencer implementation with keyboard or any other MIDI
;;input

KBRD      EQU 0FEBH
SCAN      EQU 0FEEH
MESOUT    EQU 0FF4H
TONE      EQU 0FFAH
DELAY     EQU 0FFDH

GOKEY     EQU 12H
TWRKEY    EQU 1EH
TRDKEY    EQU 1FH
PCKEY     EQU 18H
SLNT      EQU 00H

CNTRL     EQU 0F9H   ;UART CONTROL PORT ADD.
UART      EQU 0F8H   ;UART DATA PORT ADD.
MODE      EQU 4EH    ;MODE WORD=01001101B
COM       EQU 14H    ;COMMAND WORD=0XXXX100B
RESET     EQU 40H    ;RESET COMMAND

HUMPT EQU 0F4H      ;HUMKILL PORT ADDRESS

        ORG 1800H
START:
        LD SP, 1F00H
        DI
        OUT (80H), A
        LD A, 80H      ;A OUTPUT, B INPUT
        OUT (0C3H), A

PRESS:
        LD IX, SEQST
        CALL MESOUT
        CALL KBRD      ;WAIT TILL USER PRESSES A KEY
        CP TWRKEY      ;IF IT'S TWRKEY THEN GO TO RECORD
        JP Z, CH1
        CP TRDKEY      ;ELSE IF IT'S TRDKEY THEN GO TO PLAYBK
        JP Z, CH2
        JP PRESS       ;ELSE LOOP BACK

;;Routine CH1 allows user to play and record a melody
CH1: LD SP, 1F00H
      LD C, CNTRL
      LD B, 00H
      OUT (C), B      ;SEND DUMMY WORD 3 TIMES
      OUT (C), B
      OUT (C), B
      LD B, RESET     ;RESET UART
      OUT (C), B
      LD B, MODE       ;SEND MODE INSTR for baud rate=asyn x16, 8bit,
      OUT (C), B      ;no parity, 1 stop bit
```

```

LD B, COM          ;SEND COMMAND INSTR to receive enable
OUT (C), B
LD HL, INTRP      ;set up MODE 1 interrupt
LD (1F41H), HL
IM 1
LD IX, RECST
CALL MESOUT
LD IY, NRECT      ;POINT IY TO TABLE TO STORE NOTES
LD (IY+0), 0FFH  ;"END OF TABLE" MARKER
LD HL, HUMKILL    ;POINT HL TO HUMKILL status
LD A, (HL)
OUT (HUMPT), A   ;send it to HUMKILL address
LD B, 00H
EI                ;enable interrupt
LOOP: LD IX, PORT1 ;output notes to ports
LD B, (IX+2)
LD C, (IX+1)
OUT (C), B
LD IX, PORT2
LD B, (IX+2)
LD C, (IX+1)
OUT (C), B
LD IX, PORT3
LD B, (IX+2)
LD C, (IX+1)
OUT (C), B
JP LOOP          ;loop while waiting for interrupt

;;Interrupt service routine
INTRP:
DI
LD HL, 40        ;SWITCH BOUNCE
CALL DELAY
LD HL, STBYTE    ;point HL to STBYTE status
LD A, (HL)
CP 00H           ;CHECK IF STATUS BYTE RECIEVED YET
JP NZ, GETNOTE  ;IF SO, MUST BE A DATA BYTE
IN A, (UART)     ;O.W. READ IN STATUS BYTE
AND 0F0H        ;block out D3-D0
CALL CHECKSTB   ;check if it's a valid status byte
JP NZ, DONE     ;IF NOT A STATUS BYTE, IGNORE
LD (HL), A      ;O.W. SAVE STATUS BYTE
DONE: EI
RETI            ;return from interrupt

;Routine to check status byte
CHECKSTB:
CP 90H
RET Z
CP 80H
RET Z
RET

;Routine to receive data byte after status byte has been received
GETNOTE:
LD D, A         ;SAVE STATUS BYTE in D
LD HL, DTBYTE   ;POINT TO DTBYTE (note value) status

```

```

LD A, (HL)
CP 00H           ;CHECK IF DATA BYTE RECIEVED YET
JP NZ, IGNDB    ;IF SO, MUST BE 2ND DATA BYTE (velocity)
IN A, (UART)    ;O.W. READ IN 1ST DATA BYTE
SUB 24H         ;adjust enumeration to start with C=0H
LD C, A         ;save note into BC
LD B, 00H
LD (HL), 01H    ;set DTBYTE to RECIEVED
LD HL, PTRN     ;point HL to note voltages table
ADD HL, BC      ;find corresponding voltage
LD B, (HL)      ;SAVE NOTE VOLTAGE
BIT 4, D        ;TEST BIT 4 OF STATUS BYTE
JP NZ, PLAYNOTE ;IF IT'S 1, NOTE ON
CALL FINDPORT   ;O.W. IT'S NOTE OFF, FIND A PORT PLAYING THE NOTE
JP NZ, DONE     ;IF NO PORT PLAYING GIVEN NOTE, IGNORE
LD C, (IX+1)    ;O.W. GET PORT ADDRESS
LD HL, HUMKILL  ;point HL to HUMKILL status
LD A, C         ;find which port is playing the note
AND 0FH        ;BLOCK D7-D4 OF PORT ADDRESS
CP 00H         ;FIND PORT PLAYING THE NOTE
JP Z, RBIT      ;if port 0 => goto RBIT
CP 01H         ;if port 1 => goto RBIT1
JP Z, RBIT1    ;if port 1 => goto RBIT1
CP 02H         ;if port 2 => goto RBIT2
JP Z, RBIT2    ;if port 2 => goto RBIT2
                ;o.w. leave HUMKILL status as was

RBRET:
LD (IX+0), 0D   ;SET PORT STATUS TO FREE
LD (IX+2), 00H  ;SET NOTE TO 00H
JP DONE        ;return from interrupt

;;Routines to reset appropriate bits of HUMKILL to allow notes to be silenced
RBIT:
RES 0, (HL)     ;RESET BIT 0 OF HUMKILL STATUS
LD A, (HL)
OUT (HUMPT), A
JP RBRET

RBIT1:
RES 1, (HL)     ;reset BIT 1 of HUMKILL status
LD A, (HL)
OUT (HUMPT), A
JP RBRET

RBIT2:
RES 4, (HL)     ;reset BIT 4 of HUMKILL status
LD A, (HL)
OUT (HUMPT), A
JP RBRET

;;Routine to ignore velocity word and reset STBYTE and DTBYTE status
IGNDB:
IN A, (UART)    ;read in velocity byte
LD A, 00H
LD HL, STBYTE
LD (HL), A     ;RESET STBYTE TO NOT RECIEVED
LD HL, DTBYTE
LD (HL), A     ;RESET DTBYTE TO NOT RECIEVED

```

```

        JP DONE                ;return from interrupt

;;Routine to find a port playing given note
FINDPORT:
        LD IX, PORT1
        LD A, (IX+2)           ;compare note currently playing on port 1
        CP B                   ;to note in question
        RET Z                  ;if they are the same return on Z
        LD IX, PORT2
        LD A, (IX+2)           ;else compare to note playing on port 2
        CP B
        RET Z                  ;if they are the same return on Z
        LD IX, PORT3
        LD A, (IX+2)           ;else compare to note playing on port 3
        CP B
        RET Z                  ;if they are the same return on Z
        RET                    ;o.w. return with NZ

;;Routine to play out a note
PLAYNOTE:
        CALL TESTPORT         ;find a free port to play a note on
        JP NZ, DONE           ;IF NO FREE PORT AVAILABLE, IGNORE THE NOTE
        LD C, (IX+1)          ;O.W. GET PORT ADDRESS
        LD A, B                ;get note
        OUT (C), A            ;output it on the port
        LD HL, HUMKILL        ;SAVE CURRENT HUMKILL STATUS
        LD A, C
        AND 0FH               ;BLOCK B7-B4 OF PORT ADDRESS
        CP 00H                ;FIND PORT PLAYING THE NOTE
        JP Z, SBIT            ;if port 0 => goto SBIT
        CP 01H
        JP Z, SBIT1           ;if port 1 => goto SBIT1
        CP 02H
        JP Z, SBIT2           ;if port 2 => goto SBIT2
                                ;o.w. leave HUMKILL status as was

SBRET:
        LD (IX+0), 1D         ;SET PORT STATUS TO BUSY
        LD (IX+2), B          ;SET NOTE PLAYING
        LD (IY+0), B          ;SAVE NOTE VALUE
        INC IY
        LD (IY+0), C          ;SAVE PORT IT PLAYS ON
        LD (IY+1), 0FFH      ;PUT "END OF TABLE"
        INC IY
        JP DONE                ;return from interrupt

;;Routines to set appropriate bits of HUMKILL to allow notes to be heard
SBIT: SET 0, (HL)             ;SET BIT 0 OF HUMKILL STATUS
        LD A, (HL)
        OUT (HUMPT), A
        JP SBRET

SBIT1:
        SET 1, (HL)          ;SET BIT 1 OF HUMKILL STATUS
        LD A, (HL)
        OUT (HUMPT), A
        JP SBRET

SBIT2:
        SET 4, (HL)          ;SET BIT 4 OF HUMKILL STATUS

```

```

LD A, (HL)
OUT (HUMPT), A
JP SBRET

;;Routine to test which port is free to play a new note
TESTPORT:
LD IX, PORT1
LD C, (IX+0)      ;check status of port 1
LD A, C
CP 00H
RET Z             ;if it's free return on Z
LD IX, PORT2
LD C, (IX+0)      ;check status of port 2
LD A, C
CP 00H
RET Z             ;if it's free return on Z
LD IX, PORT3
LD C, (IX+0)      ;check status of port 3
LD A, C
CP 00H
RET Z             ;if it's free return on Z
RET              ;o.w. return with NZ

STBYTE:   DEFB 00H      ;STATUS WORD RECIEVED?

DTBYTE:   DEFB 00H      ;1ST DATA WORD RECIEVED?

PORT1:
DEFB 00H      ;PORT STATUS, 0-FREE, 1-BUSY
DEFB 0F0H     ;PORT ADDRESS
DEFB 00H      ;NOTE PLAYING, 00-no note

PORT2:
DEFB 00H
DEFB 0F1H
DEFB 00H

PORT3:
DEFB 00H
DEFB 0F2H
DEFB 00H

;HUMKILL STATUS
HUMKILL:
DEFB 00000000B ;D0 controls port 1, D1 controls port2,
                ;D4 controls port 3
                ;Dx=0 = port muted, Dx=1 = port playing

;;PLAYBACK routine for user-entered melody
CH2: LD HL, 250      ;TIME CONSTANT FOR DELAY
LD IY, NRECT        ;point IY to user-recorded melody table
PLAY: LD B, (IY+0)  ;GET NOTE TO PLAY
LD A, B
CP 0FFH             ;CHECK IF IT'S THE END OF THE TABLE
JP Z, START        ;IF SO, RETURN TO THE MAIN MENU
INC IY
LD C, (IY+0)        ;GET PORT TO PLAY IT ON
LD A, C

```

```

CP 0FFH           ;CHECK IF IT'S THE END OF THE TABLE
JP Z, START      ;IF SO, RETURN TO THE MAIN MENU
OUT (C), B       ;play note
CALL DELAY
INC IY
JP PLAY

```

```
PLRST:    DEFM '-SONG-'
```

```
RECST:    DEFM 'RECORD'
```

```
PLAYST:   DEFM '-PLAY-'
```

```
SEQST:    DEFM 'CHOOSE'
```

```
PREREQT:
    DEFB 0FFH
```

```

PTRN: DEFB 15D    ;NOTE=C
      DEFB 20D
      DEFB 25D
      DEFB 30D
      DEFB 35D
      DEFB 40D
      DEFB 45D
      DEFB 49D
      DEFB 54D
      DEFB 59D
      DEFB 63D
      DEFB 67D    ;NOTE=B
      DEFB 72D    ;NEXT OCTAVE
      DEFB 77D
      DEFB 82D
      DEFB 87D
      DEFB 92D
      DEFB 97D
      DEFB 102D
      DEFB 106D
      DEFB 111D
      DEFB 116D
      DEFB 120D
      DEFB 124D   ;NOTE=B
      DEFB 129D   ;NEXT OCTAVE
      DEFB 134D
      DEFB 139D
      DEFB 144D
      DEFB 149D
      DEFB 154D
      DEFB 159D
      DEFB 163D
      DEFB 168D
      DEFB 173D
      DEFB 177D
      DEFB 181D   ;NOTE=B
      DEFB 186D   ;NEXT OCTAVE
      DEFB 191D
      DEFB 196D

```

DEFB 201D  
DEFB 206D  
DEFB 211D  
DEFB 216D  
DEFB 220D  
DEFB 225D  
DEFB 230D  
DEFB 234D  
DEFB 238D

NRECT:        DEFS 1024D