Auto-Chromatic Instrument Tuner
Electrical Engineering Senior Design Project

Prepared By:

Erin M. Smith

Prepared For:

Dr. James Irwin, Senior Project Faculty Advisor
and
Dr. Winfred Anakwa, Senior Project Laboratory Coordinator
Department of Electrical and Computer Engineering and Technology
Bradley University, Peoria, IL 61625

Prepared On:

February 12, 2001

ABSTRACT

The auto-chromatic tuner is an 8031 microprocessor based device which in real time compares the pitch (note name with accidental) of a tone provided by the user with standard concert pitches. The tuner accepts an input tone from a musical instrument or voice, and determines the fundamental frequency of that waveform. The fundamental frequency is used to index a look-up table of pitches. In manual-tune mode the indexing is referenced to the pitch, and octave, set by the user while in auto-tune mode the microprocessor chooses the nearest pitch and octave. A series of LED's provide information to the user as to how close the fundamental frequency of the input tone is to the chosen concert pitch. In the audible reference pitch mode the pitch and octave are selected by the user and a square wave with this fundamental frequency is played through a speaker.

Table of Contents

I. INTRODUCTION

This project fulfills the Senior Design Project requirement for a Bachelor of Science degree in Electrical Engineering.

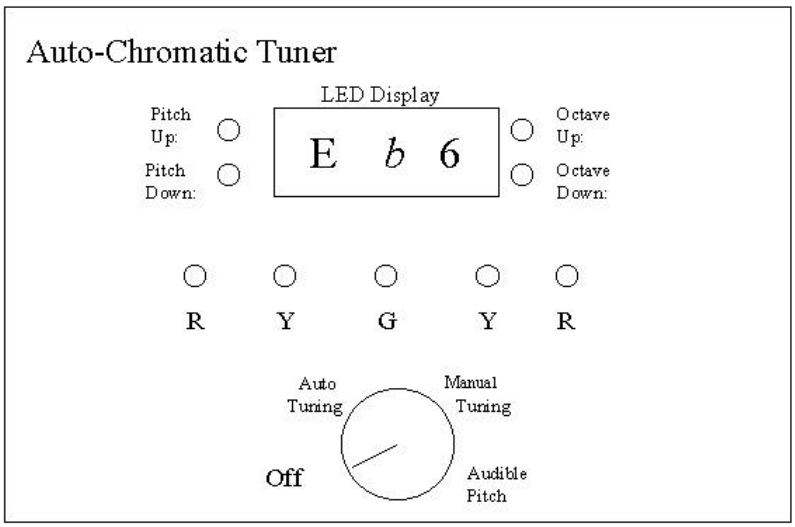
The tuning of musical instruments is important to musicians, whether playing as individuals or as part of an ensemble. For individuals, it is helpful while practicing to check the intonation of a note or multiple notes that may be out of tune. Each instrument has notes that have worse intonation relative to the rest of the notes played, which are more difficult to tune. Also, in the extreme ranges both high and low, the pitch may vary more drastically. In ensembles such as band or orchestra, the instrument tuner is important to tuning the group. An audible reference pitch may be played initially for tuning, and if the musicians are still out of tune after hearing and attempting to adjust to the reference pitch, automatic or manual tune modes may be used to tune individuals.

II. TOP-DOWN DESIGN

Top-down design was the method used for this project. This helps make the project more manageable by requiring the design engineer to look at it as a series of steps towards the overall goal of completion. A timeline listing various milestones is developed and used as a guide during the design process. It also aids in preventing integration problems in the design, avoiding complications in subsequent steps of the design process. The first step in top-down design is developing a functional description of the project.

A. Functional Description

A functional description summarizes the basic operation of the system. When read, the user should have a basic understanding of what the system does. The functional description consists of the user interface, which can be seen if Figure 1. Table 1 contains a description of each element of the user interface, as well as a description of the operating modes.



Pitches:
G: Green LED ± 5 cents
Y: Yellow LED ± 5-15 cents
R: Red LED ± 15-50 cents

Figure 1: Front Panel Diagram

In Figure 1, the diagram for the front panel can be seen. It consists of an LED display, pitch and octave up/down buttons, five LED's, and a power/mode switch. The LED display shows the note name, quality (whether it is natural or flat), and the octave. In the figure, the note displayed if E-flat in the 6^{th} octave. The pitch up/down buttons are used to run through the twelve chromatic pitches ranging from C to B. The octave up/down buttons are used to move between the eight possible octaves for tuning: zero through seven. The pitch and octave buttons are utilized in Manual Tune and Audible Reference Pitch modes only, not in Auto Tune mode. Five LED's are utilized to provide digital tuning information. In Auto Tune mode, the red LED's on the far left and far right indicate a pitch that is 15-50 cents out of tune. In Manual and Audible Reference Pitch modes, the red LED's represent a pitch which is greater than 15 cents out of tune, it could potentially indicate that a note is more than 50 cents out of range. The Power/Mode switch simply indicates if the tuner if Off or in one of the three modes: Auto Tune, Manual Tune, or Audible Reference Pitch mode.

Input	**Description**
Microphone	(Not shown) Converts the audio signal of the played pitch into an electrical signal.
Power/Mode Switch	A four position rotary switch selects one of four modes, Off, Auto Tune, Manual Tune, and Audible Reference Pitch.
Pitch Selector	Pitch and Octave Up/Down pushbuttons select the pitch (C to B, chromatically) and octave (0 to 8) to be played in Audible Reference Pitch mode, or selects the tuning pitch in Manual Tune mode.
Output	**Description**
Pitch Indicator	A three character seven segment display which indicates the pitch that the device detects in Auto Tune mode, or which indicates the selected pitch in Manual Tune and Audible Reference Pitch modes. The pitch is displayed in octave designation form (i.e. E4, B*b*2), where the first two characters are the pitch class (letter name followed by a *b* for flat), and the last character is the octave.
Digital Tuning Indicator	A set of 5 LED's to indicate fine tuning. A center green LED indicates the pitch is in tune. The two LED's around the center LED are yellow and indicate the played pitch is less than 10 cents out of tune, but not in tune. The two outer red LED's indicate the played pitch is more than 10 cents out of tune. The left red LED of each pair is lit when the pitch is flat, the right LED is lit when the pitch is played sharp.
Speaker	(Not shown) Produces the selected pitch in Audible Reference Pitch mode.
Mode	**Description**
Off	The power is off. The device consumes no electrical energy.
Auto Tuning	The device finds the closest valid pitch to the played pitch, then compares the two. The Pitch Indicator displays the closest valid pitch, and the Digital Tuning Indicator displays the tonality.
Manual Tuning	The user selects the pitch to tune. The device compares the played pitch to the selected pitch and displays the tonality with the Digital Tuning Indicator. The selected pitch is displayed on the Pitch Indicator.
Audible Reference Pitch	The user selects the pitch to be played. The Pitch Indicator displays the selected pitch, and the Speaker produces the selected pitch

Table 1: Functional Description

B. System Level Block Diagram

After creating the functional description, the next step is to develop a hardware block diagram. The block diagram for this system can be seen in Figure 2. The system level block diagram shows the general layout for the tuner, showing subsystems. It also illustrates how all of the subsystems are interconnected. The block diagram clearly shows the path the signal follows from the input to the output.

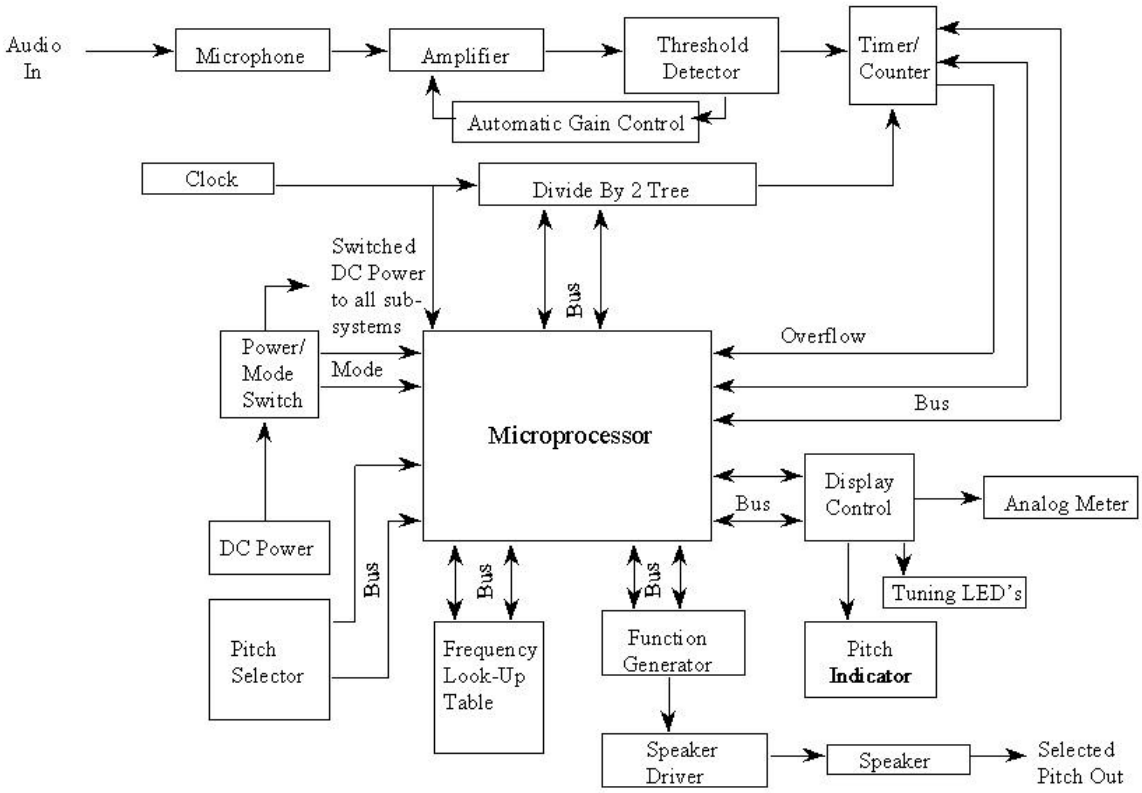


Figure 2: System Level Block Diagram

As can be seen in the block diagram, the 8031 microprocessor is central to the operation of the auto-chromatic instrument tuner. The microprocessor measures the input and then outputs the correct responses. The signal enters through the microphone and then goes through input conditioning hardware, including an amplifier, threshold detector, and Automatic Gain Control. The Timer/Counter then determines the period of the waveform, and subsequently sends this information to the microprocessor. From this

point the microprocessor determines the relative intonation of the waveform by comparing it to values in the frequency look-up table. Finally, the 8031 microprocessor will send the corresponding output values from the table to the display hardware. In order to understand how the software operates, a software flow chart is necessary.

C. Software Flow Chart

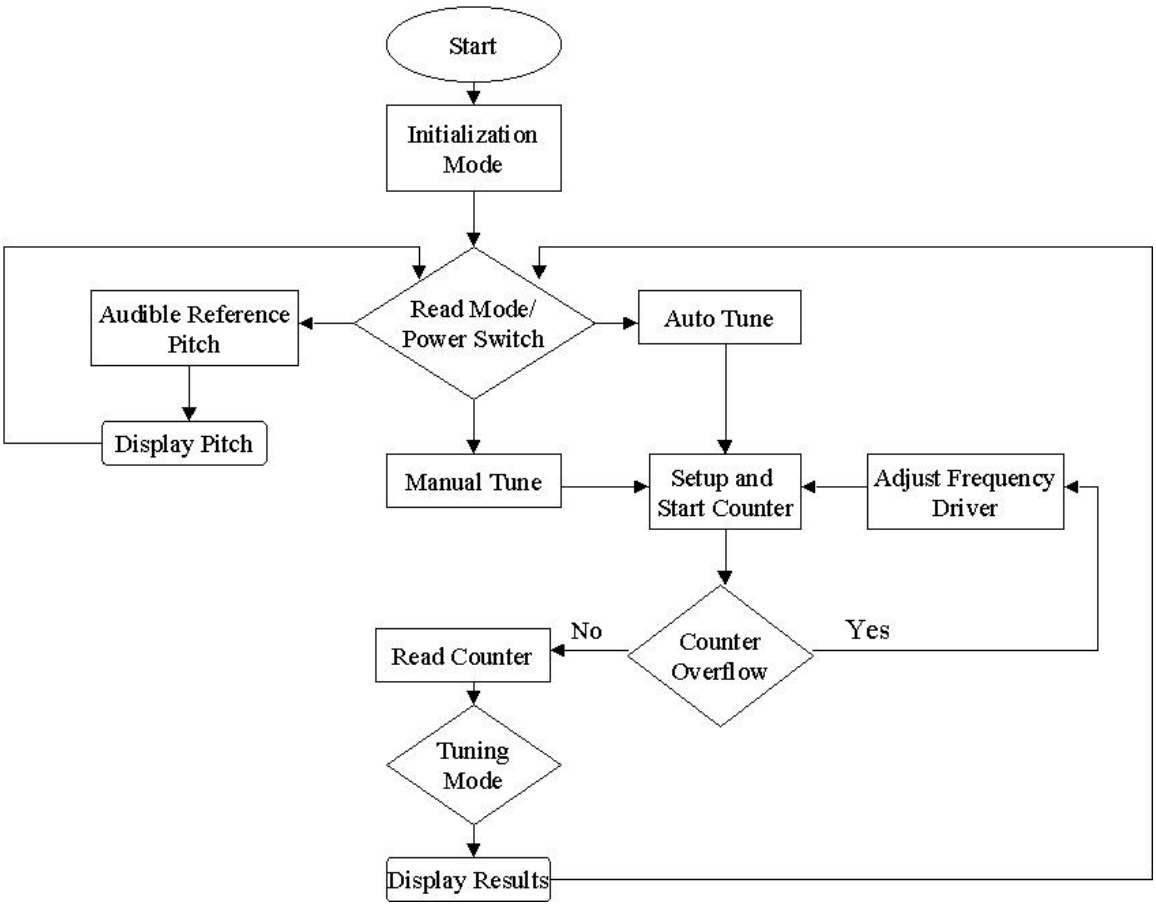


Figure 3: Software Flow Chart

Figure 3 shows the basic operation of the software written for the device. First, the Tuner enters an initialization mode. Next the correct operating mode would be chosen but since Automatic tuning mode is the only mode which is complete, the device goes directly to the Auto Tune branch of the code. Then a clock signal is counted to determine the input signal period. Next, a course of action is taken based on whether the counter overflows or not. If an overflow occurs the clock input to the counter is too high in

frequency. The frequency is lowered in divide-by-two steps until overflow does not occur. At this time, the count is read and depending upon the specific mode of operation, manual or automatic, the results are then displayed. The device is then reset to tune the next note played or enter a different mode. When Audible Reference Pitch is implemented, this mode will display the appropriate output signal and play it through the speaker.

III. THEORETICAL BACKGROUND AND INVESTIGATION

A. Musical Pitch to Frequency Equation

Since this project was a continuation from the preceding year, research was already completed to explain the relationship between the frequencies of notes in a scale. The frequencies of the pitches in the chromatic scale are related by the equation:

$$(f_1 / f_2) = 2^{(N/12)} \qquad \text{[Eq 1]}$$

where f_1 and f_2 are the frequencies of two pitches in the musical scale and N is the number of half steps between the pitches. Each pitch in the chromatic scale (containing all twelve tones) is one half-step away from the neighboring pitches.

Tuning error is calculated in cents, which is one-hundredth of the distance between neighboring pitches, in logarithmic spacing. Mathematically, this can be expressed as:

$$E_t = 100*(N/12)*\log_2 (f_t / f_r) \qquad \text{[Eq 2]}$$

where E is the tuning error in cents, f is the frequency being tuned, f is the frequency of the reference pitch, and N is the number of half steps from the reference pitch to the pitch being tuned.

B. Octave Significance

A property of musical signals which proved significant when coming up with the design for this project was that each octave is related to its adjacent octave by a multiple of two. Because of this property, the Tuner can scale every octave to the zero octave by changing the counter clock. This significantly reduces the amount of data that needs to be stored. (Instead of having a table of values for all of the octaves that can be tuned, only

one table of values is needed, which can be used for all octaves when they are divided down to the appropriate frequency range.) The table contains data which is spaced at five cent increments, since this was the original design specification. This specification was chosen by the previous group because the average human ear cannot detect differences of less than five cents.

IV. DESIGN IMPLEMENTATION

Several goals were added to the project, since the project was a continuation from a previous year. The major goals were to complete Manual Tune and Auto Tune modes, expand the digital tuning, and implement Automatic Gain Control. The schematics for the hardware of the project were not modified, and can be seen in the paper by Robert Schmanski and Craig Janus, written in 1999.

A. Manual Tune Mode

One goal was to complete the manual tune mode. From the previous year's documentation, it seemed as though not much effort was needed to complete manual tune mode because a portion of the code had already been completed. However, the portion of code for Manual Tune mode could not be located. After a great deal of looking through the existing software and learning how it operated, the conclusion was made that an excessive amount of work needed to be done in order to implement Manual Tuning mode. In addition, the software for reading the position of the Mode Switch was not implemented into the body of the final software though it had been demonstrated by itself at an earlier juncture in time. The decision was made to leave this portion of the project to future development.

B. Automatic Gain Control

Automatic Gain Control was important to increase the reliability of the threshold detector. The idea behind Automatic Gain Control is to highly amplify input signals which are very small, and minimally amplify large amplitude signals. Changing the level of amplification does not change the frequency, and thus does not effect the tuning

measurement. This keeps the voltage levels to the threshold detector more consistant which reduces false triggering.

A significant portion of time was devoted to determining a good design for Automatic Gain Control. One of the first designs involved an amplifier circuit with a varistor in the feedback loop. However, the design did not work as anticipated because a manufactured varistor could not be found which would work in the appropriate voltage range. The next design involved a gain circuit with an FET in the feedback loop. The principle of ACG did not hold in this case either, because all of the input signals remained small at the output. Finally, an article was located which explained a circuit which implements AGC using a digital potentiometer. This design can be seen in Figure 4.

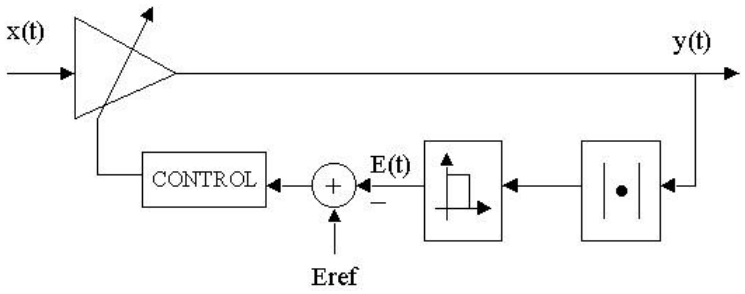


Figure 4: AGC circuit using a digital potentiometer

The above circuit works to maintain a constant energy level at the output y(t). The output y(t) goes into a full-wave rectifier followed by a filter in order to produce an estimate, E(t), of the signal energy. Next, a subtractor compares this energy signal against a preselected reference value. The difference causes the control circuit to vary the amplifier gain, which in turn keeps E(t) close to the value of E_f .

Parts for this particular design were ordered, but did not arrive in a timely manner. Therefore, the decision was made that the next project group will build and test the above design.

C. Expand Digital Tuning

Due to the fact that the analog portion of the tuner ceased to work during this portion of the project, another goal was to increase the digital tuning resolution. In order to do this, the tables contained on the GAL chips needed to be expanded considerably. The original design was for five LED's, and the eventual goal is to have seven to nine digital tuning LED's, in order to give more detailed tuning information. Time was spent expanding the tables from five LED's to nine LED's, which increased the table sizes by a factor of 16 (2^4). Although the tables have been expanded, nine LED digital tuning could not be implemented due to hardware limitations. The next project group will work to expand the hardware for nine LED's and implement a bread board design to decrease the size and wire-wrapping.

D. Auto Tune Mode

Although significant work was done on Auto Tune mode the previous year, it was not completed. Auto Tune mode only worked with pre-selected octave. Once the user selected the desired octave in which to tune, the divide-by-two circuit was set accordingly and the tables could be searched through based on the division to the zero octave, as explained previously. However, Auto Tune mode did not work without this pre-selected octave because the overflow interrupt worked was not implemented. In general when the counter overflow occurred, the divide-by-two circuit needed to be changed and thus a different octave searched. The original plan, from the previous year's report, was that the device would start from the zero octave as a base, and when overflow occurred, the octave would be incremented and the divide-by-two circuit reset. However, the count which is measured is related to the period of the waveform, which is the inverse of the frequency. Due to this fact, the original concept was backwards, it is in fact necessary to start searching from the highest octave and work down from there. Because the original process was incorrect, as modifications were made to the software, an overflow was never occurring, and thus the changes in the code appeared to make no difference. After much thought and going over the code thoroughly, the error was found. Luckily it did not require much to correct it. Each time an interrupt occurs due to overflow, the octave is decremented until an interrupt does not occur, at which point the correct tuning octave

has been reached. The count is repeated to verify its value and the tables are searched and the appropriate tuning information output to the display. This did not cure all problems. If the next note was in a higher octave the count would be incorrect. The final software was eventually implemented, which was based on always starting the search over again from the highest octave. The final version of the software can be seen in the Appendix.

V. DESIGN TESTING

Although much of the work done on this project was research and studying how to expand it further, design testing was completed for Auto Tune Mode. The tables remained the same, so the tuning information was known to be correct from thorough testing by the previous group. However, once software was completed for Auto Tune mode without the pre-selected octave, many input signals were tested for accuracy. A Hewlett-Packard function generator was utilized extensively to output various frequencies and wave shapes through a small speaker. The main trial was to see if the device was appropriately utilizing the overflow interrupt and thus performing the divide-by-two function correctly. Once the code was modified several times and this goal was reached, a variety of input signals were tested in different octaves and with varying intonations (i.e.-exactly in tune or varying degrees of sharp and flat). Also, a clarinet was played throughout the range to test how the device worked with an actual musical instrument. This method of testing proved very successful and was also useful for the demonstration of the project.

In addition, some testing was completed to determine the most desirable output waveform and duty cycle for Audible Reference Pitch. Again, a function generator and speaker were utilized in this test. After trying a variety of duty cycles and wave shapes (including square, sinusoidal, triangle, and sawtooth), it was determined that a square wave with a fifty percent duty cycle produces the most desirable sound to use as a reference pitch. This information will be used when the function generator is designed for the Audible Reference Pitch mode.

VI. CONCLUSION

Auto Tune mode was fully implemented during the course of this project. It was taken from the point of working with pre-selected octave only, to successfully searching for both the note and octave. It took a great deal of time to reach this point due to various setbacks with the device not working as it had the previous year. Research was completed on Automatic Gain Control, an AGC circuit was found, and parts were ordered. When the parts become available, the circuit can be built and tested. The realization was reached that Manual Tune mode was not simply a minor modification of Auto Tune mode. Implementing Manual Tune mode will involve creating a flow chart to write the software separate from Auto Tune mode, and eventually integrating the two together into one piece of software.

VII. APPENDIX

Final Software (Test7.a51):

```
; ***************************************************************************
; Erin Smith
; Full system test, auto tune mode only
; EE 452 Senior Project
;
;  Register Use:
;    R0, R1 - high and low bytes of 16 bit delay loop
;    R2, R3 - high and low bytes of measured period
;    R4     - table index
;    R5     - octave
;    R6     - input status
;
; Updated: 10/19/00 by Erin Smith, successful in performing auto-tune
; ***************************************************************************


TEST3:

; memory map

STARD   EQU     0000h           ; address of start of code
HSTART  EQU     1B00h           ; address of high table
LSTART  EQU     1C00h           ; address of low table
DIG     EQU     1D00h           ; address of digital tuning table
PITCH   EQU     1E00h           ; address of pitch table
INPUT   EQU     0E000h          ; address of input switches
DBT     EQU     0E800h          ; address of divide by 2^n chip
SEVSEG  EQU     0F000h          ; address of 7-seg display

; interrupt vector definitions

X0_vector       equ     0003h   ; ext 0
X1_vector       equ     0013h   ; ext 1
T0_vector       equ     000Bh   ; timer 0
T1_vector       equ     001Bh   ; timer 1
S0_vector       equ     0023h   ; serial


; main code base address
                ORG     STARD
init:           AJMP    setup

;*****************************************************************
;
;               Interrupt Jump Table
;
;*****************************************************************

                ORG     X0_vector       ; ext 0 interrupt
extint0:        SJMP    ext0srv         ; service routine
                RETI
```

```
            ORG     T0_vector       ; timer 0 interrupt
t0int:        SJMP    tmr0srv         ; service routine
            RETI

            ORG     X1_vector       ; ext 1 interrupt
extint1:      RETI                  ; disabled

            ORG     T1_vector       ; timer 1 interrupt
t1int:        RETI                  ; disabled

            ORG     S0_vector       ; UART interrupt
uartint:      RETI                  ; disabled

;************** end of interrupt jump table **********************


;*****************************************************************
;
;             Interrupt Timer 0 Service Routine
;
;*****************************************************************

tmr0srv:            CLR     TR0             ; stop timer 0
          CLR     EX0             ; disable ext int0

; timer 0 overflow has occurred, decrement octave to perform divide by 2
                    MOV     A, R5           ; get current octave
          DEC     A               ; decrement the octave
                    CJNE    A, #01h, dow    ; see if octave is one
                    MOV     A, #09h             ; if one, reset to octave 9
dow:                MOV     R5, A           ; store appropriate value in R5

; set divide by 2^n chip
          MOV     DPTR, #DBT      ; address of divide by 2^n
                    MOV     A, #80h         ; clear divide by 2^n chip
                    MOVX    @DPTR, A
          MOV     A, R5
          MOVX    @DPTR, A        ; set to divide by 2^n

          MOV     TL0, #0h        ; reset timer
          MOV     TH0, #0h

                    SETB    ET0             ; enable timer 0 ovrflw int
          SETB    EX0             ; enable ext int0
          SETB    TR0             ; start timer 0
          RETI

;*****************************************************************
;
;             External Interrupt 0 Service Routine
;
;*****************************************************************

ext0srv:      CLR     TR0             ; stop timer 0
            MOV     R2, TL0         ; get low byte of period
            MOV     R3, TH0         ; get high byte of period
```

```
            MOV     TL0, #0h        ; reset timer
            MOV     TH0, #0h
            CALL    lookup
            SETB    TR0             ; start timer 0
                    RETI



;*****************************************************************
;
;           Main Program
;
;*****************************************************************

;general 8051 initialization
setup:          MOV     SP, #70h        ; initalize stack pointer
            MOV     R0, #7Fh        ; clear 1st 128 bytes
clr_ram:        MOV     @R0, #0h        ;  of internal RAM
            DJNZ    R0, clr_ram

;user interface initialization
            MOV     DPTR, #INPUT    ; set address for inputs
            MOVX    @DPTR, A        ; clear button ffs
            MOV     P1, #0h         ; clear dig. tune LEDs

            MOV     R5, #09h        ; set to 8th octave
            CALL    dbtset          ; set divide by 2^n

;timer and interrupt initialization
            MOV     TMOD, #09h      ; set timer mode
            SETB    IT0             ; set edge triggered int
            CLR     TR0             ; stop timer 0
            MOV     TH0, #0h        ; start count at zero
            MOV     TL0, #0h

            SETB    ET0             ; enable timer 0 ovrflw int
            SETB    EX0             ; enable ext int 0

            CLR     A

            SETB    EA              ; Enable interrupt system

            SETB    TR0             ; start timer 0

            SETB    P3.2            ; activate exint0

main:
;           CALL    delay           ; button repeat delay
            SJMP    main            ; wait for interrupt

;end previously

;*****************************************************************
;
;           Table lookup and output routines
;
```

```
;*****************************************************************

lookup:
      MOV     R4,     #0h       ; clearing table index
      MOV     DPTR,   #HSTART   ; starting at high byte table
      MOV     A,      R4        ; load table index

HILOOP: MOVC    A,      @A+DPTR   ; load high byte from table
      CLR     C               ; erase borrow
      SUBB    A,      R3        ; if R3 > A, carry set
      JC      AFTER             ; then you're done with high byte
      JZ      AFTER             ; if equal, go on as well
      INC     R4              ; if not, inc table index
      MOV     A,      R4        ; load table index
      CJNE    A,      #240d,  HILOOP
                        ; do another iteration
      RET
AFTER:
      INC     DPH             ; goto low table
LOWLOOP:
      MOV     A,      R4        ; load table index
      MOVC    A,      @A+DPTR   ; load low byte from table
      CLR     C               ; erase borrow
      SUBB    A,      R2        ; if R2 > A, carry set
      JC      AFTER2            ; then you're done with low byte
      JZ      AFTER2            ; if equal, go on as well
      DEC     DPH             ; switch to high byte table
      MOV     A,      R4        ; loading high byte to compare later
      MOVC    A,      @A+DPTR
      MOV     30h,    A         ; store in internal RAM
      INC     R4              ; inc table index
      MOV     A,      R4
      CJNE    A,      #240d,  NEXT
                        ; check if outside table
      RET                 ; if so, return from routine
NEXT:                     ; else continue with compare
      MOVC    A,      @A+DPTR   ; get next high byte from table
      CJNE    A,      30h, AFTER2 ; compare two high bytes
      INC     DPH             ; return to low byte table
      AJMP    LOWLOOP           ; do another iteration

AFTER2:
      CALL    DIGTUNE
lookup_end:
      RET


;*****************************************************************
; At this point (just after AFTER2) the index, or the place of the
; breakpoint found is stored in R4. We will then use the number in
; R4 to tell us where to look in all of our succeeding tables,
; such as the DAC or the pitch indicator, and so on this next part
; we will be using these tables to find the corresponding outputs.
;*****************************************************************

; The Digital Tuning Meter will be configured as follows:
```

```
; The left red light will be lit with bit four (16), the left
; yellow light will be lit with bit three (8), the green light
; will be lit with bit two (4), the right yellow light will be
; lit with bit one (2), and the left red light will be lit with
; bit zero (1).  Remember, R4 still holds the correct place in
; the table.

DIGTUNE:
      MOV     DPTR,   #DIG      ; load start of dig table to dptr
      MOV     A,      R4        ; moving the count to A
      MOVC    A,      @A+DPTR   ; loading the correct digital tuning
                                                                    ; output
      MOV     P1,     A         ; writing the acc value to port 1
                      ; fall through to PITIND

; We have decided to use bits 4-7 to represent the octave and
; bits 0-3 to represent pitch

PITIND:
      MOV     DPTR,   #PITCH    ; load start of pitch table to dptr
      MOV     A,      R4        ; loading table index
      MOVC    A,      @A+DPTR   ; loading pitch
      MOV     R0,     A         ; store pitch
      MOV     A,      R5        ; load octave
          DEC     A             ; to normalize
      SWAP    A
      ORL     A,      R0        ; creating 8-bit octave/pitch value
      MOV     DPTR,   #SEVSEG   ; destination address
      MOVX    @DPTR,  A         ; sending info to displays
          MOV     A, #09h       ; after pitch successfully matched,
          MOV     R5, A                 ; reset octave to 9 for next search
; set divide by 2^n chip back to divide by 2^9
                MOV     DPTR, #DBT      ; address of divide by 2^n
          MOV     A, #80h       ; clear divide by 2^n chip
                MOVX    @DPTR, A
          MOV     A, R5
                MOVX    @DPTR, A        ; reset to divide by 2^9

      RET

dbtset:
      MOV     DPTR, #DBT      ; address of divide by 2^n
      MOV     A, #80h         ; clear divide by 2^n chip
      MOVX    @DPTR, A
      MOV     A, R5
      MOVX    @DPTR, A        ; set to divide by 2^n
      RET


; ***************************************************************************
;
;  delay - 2 level cascaded delay routine
;
;  Uses:
;    R0, R1
;
```

```
; ********************************************************************

delay:
      MOV     R0, #0FFh       ; initialize delay counter 1
loopB:  MOV     R1, #0FFh       ;  "                  "  2
loopA:  NOP
      DJNZ    R1, loopA
      DJNZ    R0, loopB
      RET


; Table of high byte values

ORG     HSTART

DB      245d, 244d, 244d, 243d, 242d, 242d, 241d, 240d, 239d, 239d
DB      238d, 237d, 237d, 236d, 235d, 235d, 234d, 233d, 233d, 232d
DB      231d, 231d, 230d, 229d, 229d, 228d, 227d, 227d, 226d, 225d
DB      225d, 224d, 223d, 223d, 222d, 221d, 221d, 220d, 220d, 219d
DB      218d, 218d, 217d, 216d, 216d, 215d, 214d, 214d, 213d, 213d
DB      212d, 211d, 211d, 210d, 210d, 209d, 208d, 208d, 207d, 207d
DB      206d, 205d, 205d, 204d, 204d, 203d, 202d, 202d, 201d, 201d
DB      200d, 200d, 199d, 198d, 198d, 197d, 197d, 196d, 196d, 195d
DB      194d, 194d, 193d, 193d, 192d, 192d, 191d, 190d, 190d, 189d
DB      189d, 188d, 188d, 187d, 187d, 186d, 186d, 185d, 185d, 184d
DB      183d, 183d, 182d, 182d, 181d, 181d, 180d, 180d, 179d, 179d
DB      178d, 178d, 177d, 177d, 176d, 176d, 175d, 175d, 174d, 174d
DB      173d, 173d, 172d, 172d, 171d, 171d, 170d, 170d, 169d, 169d
DB      168d, 168d, 167d, 167d, 166d, 166d, 165d, 165d, 164d, 164d
DB      163d, 163d, 162d, 162d, 161d, 161d, 161d, 160d, 160d, 159d
DB      159d, 158d, 158d, 157d, 157d, 156d, 156d, 156d, 155d, 155d
DB      154d, 154d, 153d, 153d, 152d, 152d, 152d, 151d, 151d, 150d
DB      150d, 149d, 149d, 148d, 148d, 148d, 147d, 147d, 146d, 146d
DB      145d, 145d, 145d, 144d, 144d, 143d, 143d, 143d, 142d, 142d
DB      141d, 141d, 141d, 140d, 140d, 139d, 139d, 139d, 138d, 138d
DB      137d, 137d, 137d, 136d, 136d, 135d, 135d, 135d, 134d, 134d
DB      133d, 133d, 133d, 132d, 132d, 131d, 131d, 131d, 130d, 130d
DB      130d, 129d, 129d, 128d, 128d, 128d, 127d, 127d, 127d, 126d
DB      126d, 126d, 125d, 125d, 124d, 124d, 124d, 123d, 123d, 123d

; Table of low byte values

ORG     LSTART


DB      137d, 212d, 31d, 107d, 183d, 4d, 81d, 159d, 238d, 60d
DB      140d, 220d, 44d, 125d, 206d, 32d, 115d, 198d, 25d, 109d
DB      193d, 22d, 108d, 194d, 24d, 111d, 198d, 30d, 118d, 207d
DB      40d, 130d, 220d, 55d, 146d, 238d, 74d, 167d, 4d, 97d
DB      191d, 30d, 125d, 220d, 60d, 157d, 253d, 95d, 192d, 35d
DB      133d, 232d, 76d, 176d, 20d, 121d, 223d, 68d, 171d, 17d
DB      120d, 224d, 72d, 176d, 25d, 131d, 236d, 87d, 193d, 44d
DB      152d, 4d, 112d, 221d, 74d, 184d, 38d, 148d, 3d, 114d
DB      226d, 82d, 195d, 51d, 165d, 23d, 137d, 251d, 110d, 226d
DB      86d, 202d, 62d, 179d, 41d, 159d, 21d, 140d, 3d, 122d
DB      242d, 106d, 227d, 92d, 213d, 79d, 201d, 67d, 190d, 58d
```

```
DB      181d, 49d, 174d, 43d, 168d, 37d, 163d, 34d, 160d, 31d
DB      159d, 31d, 159d, 31d, 160d, 34d, 163d, 37d, 168d, 42d
DB      173d, 49d, 181d, 57d, 190d, 66d, 200d, 77d, 211d, 90d
DB      224d, 103d, 239d, 118d, 254d, 135d, 15d, 153d, 34d, 172d
DB      54d, 192d, 75d, 214d, 98d, 238d, 122d, 6d, 147d, 32d
DB      174d, 59d, 202d, 88d, 231d, 118d, 5d, 149d, 37d, 182d
DB      70d, 215d, 105d, 250d, 140d, 31d, 177d, 68d, 216d, 107d
DB      255d, 147d, 40d, 189d, 82d, 231d, 125d, 19d, 170d, 64d
DB      215d, 110d, 6d, 158d, 54d, 207d, 103d, 0d, 154d, 51d
DB      205d, 104d, 2d, 157d, 56d, 212d, 111d, 11d, 168d, 68d
DB      225d, 126d, 28d, 186d, 88d, 246d, 148d, 51d, 210d, 114d
DB      17d, 177d, 82d, 242d, 147d, 52d, 213d, 119d, 25d, 187d
DB      94d, 0d, 163d, 70d, 234d, 142d, 50d, 214d, 123d, 32d

; Digital tuning table

ORG     DIG

DB      16d, 16d, 16d, 16d, 16d, 16d, 16d, 16d, 8d, 8d
DB      4d, 2d, 2d, 1d, 1d, 1d, 1d, 1d, 1d, 1d
DB      16d, 16d, 16d, 16d, 16d, 16d, 16d, 16d, 8d, 8d
DB      4d, 2d, 2d, 1d, 1d, 1d, 1d, 1d, 1d, 1d
DB      16d, 16d, 16d, 16d, 16d, 16d, 16d, 16d, 8d, 8d
DB      4d, 2d, 2d, 1d, 1d, 1d, 1d, 1d, 1d, 1d
DB      16d, 16d, 16d, 16d, 16d, 16d, 16d, 16d, 8d, 8d
DB      4d, 2d, 2d, 1d, 1d, 1d, 1d, 1d, 1d, 1d
DB      16d, 16d, 16d, 16d, 16d, 16d, 16d, 16d, 8d, 8d
DB      4d, 2d, 2d, 1d, 1d, 1d, 1d, 1d, 1d, 1d
DB      16d, 16d, 16d, 16d, 16d, 16d, 16d, 16d, 8d, 8d
DB      4d, 2d, 2d, 1d, 1d, 1d, 1d, 1d, 1d, 1d
DB      16d, 16d, 16d, 16d, 16d, 16d, 16d, 16d, 8d, 8d
DB      4d, 2d, 2d, 1d, 1d, 1d, 1d, 1d, 1d, 1d
DB      16d, 16d, 16d, 16d, 16d, 16d, 16d, 16d, 8d, 8d
DB      4d, 2d, 2d, 1d, 1d, 1d, 1d, 1d, 1d, 1d
DB      16d, 16d, 16d, 16d, 16d, 16d, 16d, 16d, 8d, 8d
DB      4d, 2d, 2d, 1d, 1d, 1d, 1d, 1d, 1d, 1d
DB      16d, 16d, 16d, 16d, 16d, 16d, 16d, 16d, 8d, 8d
DB      4d, 2d, 2d, 1d, 1d, 1d, 1d, 1d, 1d, 1d
DB      16d, 16d, 16d, 16d, 16d, 16d, 16d, 16d, 8d, 8d
DB      4d, 2d, 2d, 1d, 1d, 1d, 1d, 1d, 1d, 1d
DB      16d, 16d, 16d, 16d, 16d, 16d, 16d, 16d, 8d, 8d
DB      4d, 2d, 2d, 1d, 1d, 1d, 1d, 1d, 1d, 1d

; Table of pitch codes

ORG     PITCH

DB      0d, 0d, 0d, 0d, 0d, 0d, 0d, 0d, 0d, 0d
DB      0d, 0d, 0d, 0d, 0d, 0d, 0d, 0d, 0d, 0d
DB      1d, 1d, 1d, 1d, 1d, 1d, 1d, 1d, 1d, 1d
DB      1d, 1d, 1d, 1d, 1d, 1d, 1d, 1d, 1d, 1d
DB      2d, 2d, 2d, 2d, 2d, 2d, 2d, 2d, 2d, 2d
DB      2d, 2d, 2d, 2d, 2d, 2d, 2d, 2d, 2d, 2d
DB      3d, 3d, 3d, 3d, 3d, 3d, 3d, 3d, 3d, 3d
DB      3d, 3d, 3d, 3d, 3d, 3d, 3d, 3d, 3d, 3d
DB      4d, 4d, 4d, 4d, 4d, 4d, 4d, 4d, 4d, 4d
```

```
DB      4d, 4d, 4d, 4d, 4d, 4d, 4d, 4d, 4d, 4d
DB      5d, 5d, 5d, 5d, 5d, 5d, 5d, 5d, 5d, 5d
DB      5d, 5d, 5d, 5d, 5d, 5d, 5d, 5d, 5d, 5d
DB      6d, 6d, 6d, 6d, 6d, 6d, 6d, 6d, 6d, 6d
DB      6d, 6d, 6d, 6d, 6d, 6d, 6d, 6d, 6d, 6d
DB      7d, 7d, 7d, 7d, 7d, 7d, 7d, 7d, 7d, 7d
DB      7d, 7d, 7d, 7d, 7d, 7d, 7d, 7d, 7d, 7d
DB      8d, 8d, 8d, 8d, 8d, 8d, 8d, 8d, 8d, 8d
DB      8d, 8d, 8d, 8d, 8d, 8d, 8d, 8d, 8d, 8d
DB      9d, 9d, 9d, 9d, 9d, 9d, 9d, 9d, 9d, 9d
DB      9d, 9d, 9d, 9d, 9d, 9d, 9d, 9d, 9d, 9d
DB      10d, 10d, 10d, 10d, 10d, 10d, 10d, 10d, 10d, 10d
DB      10d, 10d, 10d, 10d, 10d, 10d, 10d, 10d, 10d, 10d
DB      11d, 11d, 11d, 11d, 11d, 11d, 11d, 11d, 11d, 11d
DB      11d, 11d, 11d, 11d, 11d, 11d, 11d, 11d, 11d, 11d

END
```