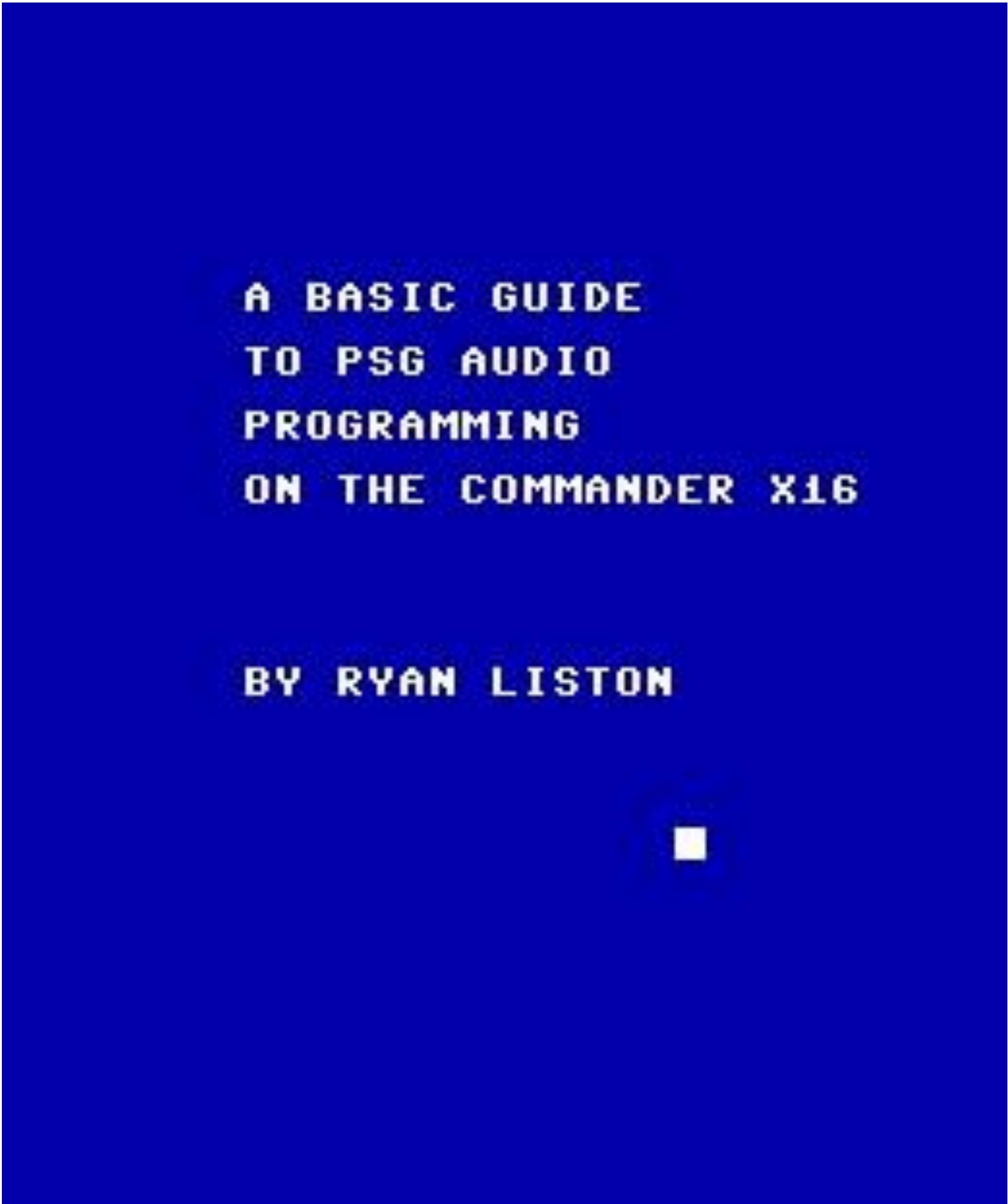
A BASIC GUIDE
TO PSG AUDIO
PROGRAMMING
ON THE COMMANDER X16
BY RYAN LISTON

A BASIC GUIDE TO PSG AUDIO PROGRAMMING ON THE COMMANDER X16

CONTENT

ABOUT X16 PSG

The programmable sound generator (PSG) in the CX16 has 16 voices located increments of 4 starting at $1F9C0 in the VERA module. That is VERA bank $1,$F9C0 or 1, 63936. The 16 voices each have 4 control registers a piece. Each voice has its own independent volume, pan and pulse width controls and is capable of 4 waveforms.

The control registers are

0:frequency word (low byte)

1:frequency word (high byte)

2:bit 0-5 volume bit 6&7 pan

3:bit 0-5 pulse width 6&7 wave form

The formula to calculate each voices memory location can be calculated with

63936+([voice]*4).

The formula to calculate the memory location for each voices control registers

63936+([voice]*4)+[control register].

*- see the chart "x16 psg register map" for a full listing of voice and control addresses.

FREQUENCY WORD is the 16 bit value you use to play a note with the PSG. The formula for calculating the frequency word is

[frequency word] = [hertz]/(48828.124/(2^17))

Control register 0 is the frequency word low byte. The formula to calculate the low byte is

int([frequency word]= -([high byte]*256))

Control register 1 is the frequency word high byte. The formula to calculate the high byte is

int([frequency word]/256)

*- See "x16 psg note chart" for a full listing of note and frequency word values.

Control register 2 contains pan and volume controls. Bits 0 to 5 controls volume. bits 6 and 7 control pan. Volume (0 = silent,
63 = loudest). Bit 6 controls right and 7 controls left with 0 being off and 1 being on. The formula for register 2 input is

([left pan]*128)+([right pan]*64)+[volume]

Control register 3 contains pulse width and wave form controls. Bits 0 to 5 control pulse width. bits 6 and 7 control wave form. Pulse width (0 = narrow , 63 = wide). Wave form (0=pulse, 1=saw, 2=triangle, 3=noise). The formula for register 3 input is

([wave form]*64)+[pulse width]

X16 PSG REGISTER MAP

frequency word = [hertz] / 48828.125 / (2^17) , high byte = int ([freq word] / 256)
low byte = int ([freq word] - [high byte])

pan / volume : bits 0 - 5 = volume (val 0 to 63) , bit 6 = right pan (0 = off , 1 = on)
bit 7 = left pan (0 = off , 1 = on)
formula for pan / volume control = (([left pan] × 128)+ ([right pan] × 64) + volume)

wave form / pulse width : bits 0 - 5 = pulse widtg (val 0 to 63)
bit 6 - 7 = wave form (0 = pulse , 1 = saw , 2 = triangle , 3 = noise)
formula for wave form / pulse width is = ([wave form] × 192) + [pulse width])

voice	controls	hex	dec	voice	controls	hex	dec
0	freq word low byte	F9C0	63936	8	freq word low byte	F9E0	63968
	freq word high byte	F9C1	63937		freq word high byte	F9E1	63969
	pan / volume	F9C2	63938		pan / volume	F9E2	63970
	wave form / pulse width	F9C3	63939		wave form / pulse width	F9E3	63971
1	freq word low byte	F9C4	63940	9	freq word low byte	F9E4	63972
	freq word high byte	F9C5	63941		freq word high byte	F9E5	63973
	pan / volume	F9C6	63942		pan / volume	F9E6	63974
	wave form / pulse width	F9C7	63943		wave form / pulse width	F9E7	63975
2	freq word low byte	F9C8	63944	10	freq word low byte	F9E8	63976
	freq word high byte	F9C9	63945		freq word high byte	F9E9	63977
	pan / volume	F9CA	63946		pan / volume	F9EA	63978
	wave form / pulse width	F9CB	63947		wave form / pulse width	F9EB	63979
3	freq word low byte	F9CC	63948	11	freq word low byte	F9EC	63980
	freq word high byte	F9CD	63949		freq word high byte	F9ED	63981
	pan / volume	F9CE	63950		pan / volume	F9EE	63982
	wave form / pulse width	F9CF	63951		wave form / pulse width	F9EF	63983
4	freq word low byte	F9D0	63952	12	freq word low byte	F9F0	63984
	freq word high byte	F9D1	63953		freq word high byte	F9F1	63985
	pan / volume	F9D2	63954		pan / volume	F9F2	63986
	wave form / pulse width	F9D3	63955		wave form / pulse width	F9F3	63987
5	freq word low byte	F9D4	63956	13	freq word low byte	F9F4	63988
	freq word high byte	F9D5	63957		freq word high byte	F9F5	63989
	pan / volume	F9D6	63958		pan / volume	F9F6	63990
	wave form / pulse width	F9D7	63959		wave form / pulse width	F9F7	63991
6	freq word low byte	F9D8	63960	14	freq word low byte	F9F8	63992
	freq word high byte	F9D9	63961		freq word high byte	F9F9	63993
	pan / volume	F9DA	63962		pan / volume	F9FA	63994
	wave form / pulse width	F9DB	63963		wave form / pulse width	F9FB	63995
7	freq word low byte	F9DC	63964	15	freq word low byte	F9FC	63996
	freq word high byte	F9DD	63965		freq word high byte	F9FD	63997
	pan / volume	F9DE	63966		pan / volume	F9FE	63998
	wave form / pulse width	F9DF	63967		wave form / pulse width	F9FF	63999

X16 PSG NOTE CHART											
Note	Hz	freq word	hex	high byte	low byte	Note	Hz	freq word	hex	high byte	low byte
A 0	27.5	74	004A	0	74	F 4	349.23	937	03A9	3	169
A# 0	29.135	78	004E	0	78	F# 4	369.99	993	03E1	3	225
B 0	30.868	83	0053	0	83	G 4	392	1052	041C	4	28
C 1	32.703	88	0058	0	88	G# 4	415.31	1115	045B	4	91
C# 1	34.648	93	005D	0	93	A 4	440	1181	049D	4	157
D 1	36.708	99	0063	0	99	A# 4	466.16	1251	04E3	4	227
D# 1	38.891	104	0068	0	104	B 4	493.88	1326	052E	5	46
E 1	41.203	111	006F	0	111	C 5	523.25	1405	057D	5	125
F 1	43.654	117	0075	0	117	C# 5	554.37	1488	05D0	5	208
F# 1	46.249	124	007C	0	124	D 5	587.33	1577	0629	6	41
G1	48.999	132	0084	0	132	D# 5	622.25	1670	0686	6	134
G# 1	51.913	139	008B	0	139	E 5	659.26	1770	06EA	6	234
A 1	55	148	0094	0	148	F 5	698.46	1875	0753	7	83
A# 1	58.271	156	009C	0	156	F# 5	739.99	1986	07C2	7	194
B 1	61.735	166	00A6	0	166	G 5	783.99	2105	0839	8	57
C 2	65.406	176	00B0	0	176	G# 5	830.61	2230	08B6	8	182
C# 2	69.296	186	00BA	0	186	A 5	880	2362	093A	9	58
D 2	73.416	197	00C5	0	197	A# 5	932.33	2503	09C7	9	199
D# 2	77.782	209	00D1	0	209	B 5	987.77	2652	0A5C	10	92
E 2	82.407	221	00DD	0	221	C 6	1046.5	2809	0AF9	10	249
F 2	87.307	234	00EA	0	234	C# 6	1108.7	2976	0BA0	11	160
F# 2	92.499	248	00F8	0	248	D 6	1174.7	3153	0C51	12	81
G 2	97.999	263	0107	1	7	D# 6	1244.5	3341	0D0D	13	13
G# 2	103.83	279	0117	1	23	E 6	1318.5	3539	0DD3	13	211
A 2	110	295	0127	1	39	F 6	1396.9	3750	0EA6	14	166
A# 2	116.54	313	0139	1	57	F# 6	1480	3973	0F85	15	133
B 2	123.47	331	014B	1	75	G 6	1568	4209	1071	16	113
C 3	130.81	351	015F	1	95	G# 6	1661.2	4459	116B	17	107
C# 3	138.59	372	0174	1	116	A 6	1760	4724	1274	18	116
D 3	146.83	394	018A	1	138	A# 6	1864.7	5005	138D	19	141
D# 3	155.56	418	01A2	1	162	B 6	1975.5	5303	14B7	20	183
E 3	164.81	442	01BA	1	186	C 7	2093	5618	15F2	21	242
F 3	174.61	469	01D5	1	213	C# 7	2217.5	5952	1740	23	64
F# 3	185	497	01F1	1	241	D 7	2349.3	6306	18A2	24	162
G 3	196	526	020E	2	14	D# 7	2489	6681	1A19	26	25
G# 3	207.65	557	022D	2	45	E 7	2637	7079	1BA7	27	167
A 3	220	591	024F	2	79	F 7	2793.8	7500	1D4C	29	76
A# 3	233.08	626	0272	2	114	F# 7	2960	7946	1F0A	31	10
B 3	246.94	663	0297	2	151	G 7	3136	8418	20E2	32	226
C 4	261.63	702	02BE	2	190	G# 7	3322.4	8919	22D7	34	215
C# 4	277.18	744	02E8	2	232	A 7	3520	9449	24E9	36	233
D 4	293.67	788	0314	3	20	A# 7	3729.3	10011	271B	39	27
D# 4	311.13	835	0343	3	67	B 7	3951.1	10606	296E	41	110
E 4	329.63	885	0375	3	117	C 8	4186	11237	2BE5	43	229

TYPE-INS

I have provided 4 simple type-in Basic programs to help you get familiar with programing the CX16s PSG. There is a simple sound generator, sound fx maker, music player and a program that turns your game controller into a musical instrument.

Feel free to play with the code. All control parameters are at the beginning of the code. This is to make it easy to run, stop, change parameters and run again to see how different values will change the sound.

When typing in the code only type lines starting with numbers. Lines starting with ! are comments and are put there to help describe the code.

You do not have to type every space. I used extra spacing to make the code easier to read. Basic does not pay any attention to spaces unless they are in between quotation marks.

Sound Generator

```
!-=============================================
!-
!- PROGRAM : PLAY SOUND
!- BY : RYAN LISTON
!- DATE : MARCH 2020
!-
!---------------------------------------------
!-
!- Plays a single sound
!-
!---------------------------------------------
!-
!- vr = voice root ( root voice register )
!- ro = voice register 0 ( freq word low byte )
!- r1 = voice register 1 ( freq word high byte )
!- r2 = voice register 2 ( pan / volume )
!- r3 = voice register 3 ( wave form / pulse
!-      width )
!- lb = frequency word low byte ( 0 to 255 )
!- hb = frequency word high byte (0 to 255 )
!- pl = pan left ( 0 = off, 1 = on)
!- pr = pan right ( 0 = off, 1 = on)
!- vl = volume ( 0 to 63 )
!- wf = wave form ( 0 = pulse , 1 = saw ,
!-      2 = triangle, 3 = square )
!- pw = pulse width ( 0 to 63 )
!- sl = sound length
!-
!---------------------------------------------

!- declare variables
!- assigns voice registers

10 vr= $f9c0 : ro = vr : r1 = vr + 1
15 r2 = vr + 2 : r3 = vr + 3

!- sets up voice sound parameters

!- change values to alter sound and tone
```

```
20 lb = 190 : hb = 2 : pl = 1 : pr = 1 : vl = 63
: wf = 0 : pw = 32 : 21 sl = 20000

!-----------------------------------------------

!- sets sound parameters to voice registers
!- sets pan / volume

30 vpoke 1 , r2 , ((pl * 128) + (pr * 64) + vl)

!- sets wave form / pulse width

40 vpoke 1 , r3 , ((wf * 64) + pw)

!-----------------------------------------------

!- plays sound

50 vpoke 1 , ro , lb : vpoke1 , r1 , hb

!-----------------------------------------------

!- timer

60 for t = 0 to sl : next t

!-----------------------------------------------

!- turns sound off

70 vpoke 1 , ro , 0 : vpoke 1 , r1 , 0

!-----------------------------------------------

!- end program

80 end

!-===============================================
```

Sound Effect Generator

```
!-=============================================
!- PROGRAM : SOUND FX 1
!- BY : RYAN LISTON
!- DATE : MARCH 2020
!-
!---------------------------------------------
!-
!- Plays a sound effect.
!-
!---------------------------------------------
!-
!- vr = voice root ( root voice register )
!- r(0) = voice register 0 ( freq word low
!-        byte )
!- r(1) = voice register 1 ( freq word high
!-        byte )
!- r(2) = voice register 2 ( pan / volume )
!- r(3) = voice register 3 ( wave form / pulse
!-        width )
!- lb = frquency word low byte ( 0 to 255 )
!- hb = frequency word high byte (0 to 255 )
!- pn = pan ( 0 = off, 1 = right on ,
!-      2 = left on ), 3 = left and right on )
!- vl = volume ( 0 to 63 )
!- wf = wave form ( 0 = pulse , 1 = saw ,
!-      2 = triangle, 3 = square )
!- pw = pulse width ( 0 to 63 )
!- sc = slice count
!- sl = slice length
!- fx(x,y) = array to hold sfx data for
!-           program use
!-
!---------------------------------------------

!- sets up variables and arrays
!- sets up sfx array fx(x,y), x = sound slice
!- y = control
```

```
10 read sc : dim fx ( sc + 1 , 5 )
15 for x = 0 to sc
20 read lb: read hb: read pn: read vl: read wf:
25 read pw: read sl
30 fx ( x , 0 ) = lb : fx ( x , 1 ) = hb
35 fx ( x , 2 ) = ( pn * 64 ) + vl
40 fx ( x , 3 ) = ( wf * 64 ) + pw
45 fx ( x , 4 ) = sl : next x : restore

!- sets up voice root  and array for voice
!- control

50 vr = 63936 : dim r ( 4 ) : for x = 0 to 3
55 r ( x ) = vr + x : next x

!-----------------------------------------------

!- play sound fx

60 for x = 0 to sc : for y = 0 to 3
65 vpoke 1 , r ( y ) , fx ( x , y ) : next y
70 for t = 0 to fx ( x , 4) : next t :  next x


!-----------------------------------------------

!- clears voice registers and ends sound and
program

80 for x = 0 to 3 : vpoke 1 , r ( x ) , 0
85 next x : end

!-----------------------------------------------

!- data table for sfx
!- slice count

90 data 15

!- each line of data is 1 slice of the sfx
```

```
!-       lb     hb   pn  vl    wf  pw    sl

100 data $00 , $02 , 1 , $20 , 1 , $00 , 329
110 data $40 , $02 , 2 , $20 , 1 , $04 , 319
120 data $80 , $02 , 1 , $20 , 1 , $08 , 309
130 data $d0 , $02 , 2 , $28 , 1 , $0d , 299
140 data $00 , $03 , 1 , $28 , 1 , $10 , 299
150 data $40 , $03 , 1 , $28 , 1 , $14 , 299
160 data $80 , $03 , 2 , $30 , 1 , $18 , 279
170 data $d0 , $03 , 2 , $30 , 1 , $1d , 279
180 data $00 , $04 , 1 , $30 , 1 , $20 , 279
190 data $40 , $04 , 1 , $38 , 1 , $24 , 279
200 data $80 , $04 , 2 , $38 , 1 , $28 , 269
210 data $d0 , $04 , 2 , $38 , 0 , $2d , 269
220 data $00 , $05 , 3 , $38 , 0 , $30 , 389
230 data $40 , $05 , 3 , $30 , 0 , $34 , 529
240 data $80 , $05 , 3 , $30 , 0 , $38 , 759
250 data $d0 , $05 , 3 , $30 , 0 , $3d , 1589

!-============================================
```

Musical Game Controller

```
!-=============================================
!-
!- PROGRAM : MUSICAL GAME CONTROLLER
!- BY : RYAN LISTON
!- DATE : MARCH , 2020
!-
!---------------------------------------------
!-
!- Plays music with NES controller or computer
!- keyboard.
!- Right = C4 , Left = D4 , Up = E4 , Down = f4
!-                               Start/Enter = G4
!- Select/Space = A4 , B/Alt = B4 , A/Ctrl = C5
!-
!---------------------------------------------
!-
!- vr = voice root ( root voice register )
!- r(0) = voice register 0 ( freq word low
!-        byte )
!- r(1) = voice register 1 ( frequency word high
!-        byte )
!- r(2) = voice register 2 ( pan / volume )
!- r(3) = voice register 3 ( wave form / pulse
!-        width )
!- fw(hb,lb) = frequency word ( high byte , low
!-             byte )
!- hb = frequency word high byte
!- lb = frequency word low byte
!- pn = pan
!- vl = volume
!- wf = wave form
!- pw = pulse width
!- ci = control input
!-
!---------------------------------------------

!- data set : 1 silent and 8 notes in 16 bit
!- frequency word value
```

```
!- leave first value as $0000 for silence

!-       rest   c4    d4    e4

10 data $0000,$02be,$0314,$0375

!-        f4    g4    a4    b4    c5

15 data $03a9,$041c,$049d,$052e,$057d

!-----------------------------------------------

!- set up for variables for voice controls

20 vr = $f9c0 : pn = 3 : vl = 63
25 wf = 1 : pw = 32 : nl = 2000 : ci = 0

!- sets up voice controls

30 dim r(4) : r(0) = vr : r(1) = r(0) + 1
35 R(2) = R(1) + 1 : R(3) = R(2) + 1

!- sets up note table array from data set

40 dim fw(9,2) : for t = 0 to 8 : read a
50 fw(t,1) = int(a / 256)
55 fw(t,2) = int(a - (fw(t,1) * 256))
60 next t : restore

!- set voice registers

70 vpoke 1 , r(2) , (pn * 64) + VL
75 vpoke 1 , R(3) , (WF * 64) + PW

!-----------------------------------------------

!- gets controller input

80 ci = joy(1)
90 if ci = 0 then 190
```

```
100 if ci = 1 then 190
110 if ci = 2 then 190
120 if ci = 4 then ci = 3 : goto 190
130 if ci = 8 then ci = 4 : goto 190
140 if ci = 16 then ci = 5 : goto 190
150 if ci = 32 then ci = 6 : goto 190
160 if ci = 64 then ci = 7 : goto 190
170 if ci = 128 then ci = 8 : goto 190
180 ci = 0

!- plays note

190 vpoke 1 , r(0) , fw(ci,2)
195 vpoke 1 , r(1) , fw(ci,1)

!- loops

200 goto 80

!-=============================================
```

Music Player

```
!-=============================================
!-
!- PROGRAM : PSG MUSIC PLAER
!- BY : RYAN LISTON
!- DATE : MARCH , 2020
!-
!---------------------------------------------
!-
!- A SIMPLE PSG MUSIC PLAYER
!-
!---------------------------------------------
!-
!- nd = note duration
!- nt = note timer
!- v to v3 = root registers for voice 0 to 3
!- h to h3 = frequency root low bytes for
!-           voices 0 to 3
!- l to t3 = frequency root high bytes for
!-           voices 0 to3
!-
!---------------------------------------------

!- declare constants and variables

10 nd = 10 : nt = 0 : v = $f9c0 : v1= v + 4
15 v2 = v1 + 4
15 v3 = v2 + 4

!---------------------------------------------

!- voice setup

20 vpoke 1, v+2, %11111111
25 vpoke 1, v+3, %11011111
30 vpoke 1, v1+2, %11100011
35 vpoke 1, v1+3, %00111111
40 vpoke 1, v2+2, %11111110
45 vpoke 1, v2+3, %01001111
50 vpoke 1, v3+2, %11111100
```

```
55 vpoke 1, v3+3, %01000011

!-----------------------------------------------

!- start of player loop

!-----------------------------------------------

!- timer

60 if ti < nt then 60

70 nt= ti + nd

!-----------------------------------------------

!- reads note values (frequency word high/
!- low bytes) from data

80 read l : read h : read l1 : readh1 : read l2 :
85 read h2 : read l3 : 85 read h3

!-----------------------------------------------




!- test for end of song.
!- if h = -1 then restore data set and restart
!-        song

90 if l = -1 then restore : goto 80

!-----------------------------------------------

!- plays notes (high byte : low byte)

100 vpoke 1 , v , l : vpoke 1 , v + 1 , h
110 vpoke 1 , v1 , l1 : vpoke 1 , v1 + 1 , h1
120 vpoke 1 , v2 , l2 : vpoke 1 , v2 + 1 , h2
```

```
130 vpoke 1 , v3 , l3 : vpoke 1 , v3 + 1 , h3

!-----------------------------------------------

!- loop

140 goto 60

!-----------------------------------------------

!-song data

!- (l,h)  v0       v1      v2       v3

150 data $b0,$00, $d3,$0d, $bf,$02, $b0,$00
160 data $00,$00, $d3,$0d, $00,$00, $00,$00
170 data $00,$00, $d3,$0d, $1c,$04, $b0,$00
180 data $00,$00, $d3,$0d, $00,$00, $00,$00
190 data $1c,$04, $a6,$0e, $75,$03, $b0,$00
200 data $00,$00, $a6,$0e, $75,$03, $00,$00
210 data $00,$00, $a6,$0e, $00,$00, $b0,$00
220 data $00,$00, $a6,$0e, $00,$00, $00,$00

230 data $b0,$00, $51,$0c, $00,$00, $b0,$00
240 data $00,$00, $51,$0c, $00,$00, $00,$00
250 data $00,$00, $d3,$0d, $00,$00, $b0,$00
260 data $00,$00, $d3,$0d, $00,$00, $00,$00
270 data $1c,$04, $a6,$0e, $00,$00, $b0,$00
280 data $00,$00, $a6,$0e, $00,$00, $00,$00
290 data $00,$00, $a6,$0e, $00,$00, $b0,$00
300 data $00,$00, $a6,$0e, $00,$00, $00,$00

310 data $b0,$00, $d3,$0d, $75,$03, $c5,$00
320 data $00,$00, $d3,$0d, $00,$00, $00,$00
330 data $b0,$00, $d3,$0d, $1c,$04, $c5,$00
340 data $00,$00, $d3,$0d, $1c,$04, $00,$00
350 data $1c,$04, $a6,$0e, $75,$03, $c5,$00
360 data $00,$00, $a6,$0e, $9d,$04, $00,$00
370 data $00,$00, $a6,$0e, $00,$00, $00,$00
380 data $00,$00, $a6,$0e, $00,$00, $00,$00
```

```
390 data $b0,$00, $d3,$0d, $1c,$04, $c5,$00
400 data $00,$00, $d3,$0d, $1c,$04, $00,$00
410 data $b0,$00, $d3,$0d, $1c,$04, $c5,$00
420 data $00,$00, $d3,$0d, $00,$00, $00,$00
430 data $1c,$04, $a6,$0e, $00,$00, $c5,$00
440 data $00,$00, $a6,$0e, $00,$00, $00,$00
450 data $00,$00, $a6,$0e, $00,$00, $00,$00
460 data $00,$00, $a6,$0e, $00,$00, $00,$00

470 data $b0,$00, $00,$00, $bf,$02, $b0,$00
480 data $00,$00, $00,$00, $00,$00, $00,$00
490 data $00,$00, $00,$00, $1c,$04, $b0,$00
500 data $00,$00, $00,$00, $00,$00, $00,$00
510 data $1c,$04, $a6,$0e, $75,$03, $b0,$00
520 data $00,$00, $00,$00, $75,$03, $00,$00
530 data $00,$00, $00,$00, $00,$00, $b0,$00
540 data $00,$00, $00,$00, $00,$00, $00,$00

550 data $b0,$00, $00,$00, $00,$00, $b0,$00
560 data $00,$00, $51,$0c, $00,$00, $00,$00
570 data $00,$00, $00,$00, $00,$00, $b0,$00
580 data $00,$00, $a6,$0e, $00,$00, $00,$00
590 data $1c,$04, $00,$00, $00,$00, $b0,$00
600 data $00,$00, $51,$0c, $00,$00, $00,$00
610 data $1c,$04, $00,$00, $00,$00, $b0,$00
620 data $00,$00, $d3,$0d, $00,$00, $00,$00

630 data $b0,$00, $1c,$04, $0e,$02, $07,$01
640 data $00,$00, $1c,$04, $00,$00, $07,$01
650 data $00,$00, $1c,$04, $00,$00, $07,$01
660 data $00,$00, $00,$00, $97,$02, $00,$00
670 data $b0,$00, $9d,$04, $97,$02, $07,$01
680 data $00,$00, $00,$00, $00,$00, $00,$00
690 data $b0,$00, $a9,$04, $00,$00, $07,$01
700 data $00,$00, $00,$00, $00,$00, $00,$00

710 data $1c,$04, $1c,$04, $0e,$02, $07,$01
720 data $00,$00, $1c,$04, $00,$00, $00,$00
730 data $00,$00, $a9,$03, $97,$02, $07,$01
```

```
740 data $00,$00, $00,$00, $97,$02, $00,$00
750 data $b0,$00, $9d,$04, $14,$03, $07,$01
760 data $00,$00, $9d,$04, $00,$00, $0e,$02
770 data $1d,$04, $2e,$05, $be,$02, $07,$01
780 data $00,$00, $2e,$05, $be,$02, $00,$00

790 data -1,0,0,0,0,0,0,0

!-============================================
```