

```

1 ; CpS 230 Team Project: Nathan Collins and Stephen Sidwell
2 ;-----
3 ; description goes here
4 ;-----
5 bits 16
6
7 extern functionThatKeepsStuffFromBreaking_
8 extern moveBlock0_
9 extern moveBlock1_
10 extern moveBlock2_
11 extern pal_counter_
12
13 ;For bootstrapped programs, all addresses start at 0
14 ;org 0x0
15
16 ; Where to find the INT 8 handler vector within the IVT [interrupt vector table]
17 IVT8_OFFSET_SLOT equ 4 * 8 ; Each IVT entry is 4 bytes; this is the 8th
18 IVT8_SEGMENT_SLOT equ IVT8_OFFSET_SLOT + 2 ; Segment after Offset
19
20
21 global start_
22
23 section .text
24 start_:
25
26 ;Make code and data segments the same to simplify addressing
27 mov ax, cs
28 mov ds, ax
29
30 ; Set ES=0x0000 (segment of IVT)
31 mov ax, 0x0000
32 mov es, ax
33
34 ; Set VGA graphics mode (320x200x8-bit)
35 mov ah, 0
36 mov al, 0x13
37 int 0x10
38
39 ;do stuff
40 jmp bootstrap
41
42 ;Task to move a block across the screen
43 task1:
44 ;External C Function
45 ;See test.c
46 call moveBlock1_
47
48 mov cx, 50000
49
50 .loop:
51 dec cx
52 jnz .loop
53
54
55 jmp task1
56
57 ;Task to move another block across the screen
58 task2:
59 ;External C Function
60 ;See test.c
61 call moveBlock2_
62
63 mov cx, 50000
64
65 .loop:
66 dec cx
67 jnz .loop
68
69 jmp task2

```

```

70
71 ;Task to keep a palette offset counting
72 ;Used to shift the color of the mandlebrot design only
73 task3:
74     ;External C Function
75     ;See test.c
76     call pal_counter_
77
78     mov     cx, 50000
79
80 .loop:
81     dec     cx
82     jnz     .loop
83
84     jmp     task3
85
86 ;IVT 8 Handler Function
87 ;Used for preemptive multitasking
88 ;Also used to play the background music
89 yield:
90     ;Flags, CS, and IP should all have been pushed by the interrupt
91     ;Push GPRs
92     pusha
93     ;Push DS and ES
94     push    ds
95     push    es
96
97     ;Assembly function to do music things
98     call    playMusic
99
100    ;Get current stack index and adjust for word size
101    mov     ax, [stack_idx]
102    mov     cx, 2
103    imul    cx
104    mov     bx, ax
105    ;Save current stack pointer
106    mov     [stacks + bx], sp
107
108    ;Check to see if the stack index needs to wrap to the first task
109    cmp     word[stack_idx], 3
110    je      .wrap
111    ;If not just increment the index
112    inc     word[stack_idx]
113    jmp     .end
114 ;if so reset index back to 0
115 .wrap:
116     mov     word[stack_idx], 0
117 .end:
118     ;Get new stack index and adjust for word size
119     mov     ax, [stack_idx]
120     imul    cx
121     mov     bx, ax
122     ;Move new stack pointer into sp
123     mov     sp, [stacks + bx]
124     pop     es
125     pop     ds
126     popa
127     ;Chain to next interrupt handler
128     jmp     far [cs:ivt8_offset] ; Use CS as the segment here, since who knows what DS
                                   is now
129
130 ;Kickstart function to start the first task3
131 ;Clears the initial stack and in the process sets the interrupt flag
132 start_first_task:
133     pop     es
134     pop     ds
135     popa
136     iret
137

```

```

138 bootstrap:
139     ;Pay no attention to the order of this code
140
141     ;Set up stacks for all 4 tasks
142     ;Same steps are taken for all 4
143
144
145     ;Moving block 1
146     ;Move the task's stack pointer into sp
147     mov     sp, stack2 + 255 ; top of stack2
148     ;Set up initial stack by pushing flags, cs, code address of the task, ds, and cs
149     pushf
150     push     cs
151     push     task2             ; location to return to
152     pusha
153     push     ds
154     push     es
155     ;Save the task's stack pointer into the stack pointer array
156     mov     [stacks + 2*1], sp
157
158     ;Moving block 2
159     ;See the above setup for more comments
160     mov     sp, stack1 + 255 ; top of stack1
161     pushf
162     push     cs
163     push     task1             ; location to return to
164     pusha
165     push     ds
166     push     es
167     mov     [stacks + 2*0], sp
168
169     ;Task to change color pallete
170     ;See the first setup for more comments
171     mov     sp, stack4 + 255 ; top of stack4
172     pushf
173     push     cs
174     push     task3             ; location to return to
175     pusha
176     push     ds
177     push     es
178     mov     [stacks + 2*3], sp
179
180
181     ;Mandlebrot setup
182     ;Same as all the above with one exception
183     ;The stack pointer is not stored or changed after the setup is complete
184     mov     sp, stack3 + 255 ; top of stack1
185     pushf
186     push     cs
187     push     mandlebrot_task    ; location to return to
188     pusha
189     push     ds
190     push     es
191
192
193
194
195     ;Install IVT 8 handler for task switching
196     cli
197     mov     ax, [es:IVT8_SEGMENT_SLOT]
198     mov     [ivt8_segment], ax
199     mov     ax, [es:IVT8_OFFSET_SLOT]
200     mov     [ivt8_offset], ax
201     mov     [es:IVT8_SEGMENT_SLOT], cs
202     mov     word [es:IVT8_OFFSET_SLOT], yield
203
204     ;Since the stack pointer is still on the mandlebrot_task's stack that will be the
205     ;first task exeuctuted
206     ;Kickstarts the task outside of the interrupt handler

```

```

206     jmp     start_first_task
207
208     ;
-----
209     ; now all the really gross code for the music
210
211     ; random numbers copied from the example code.
212     SPEAKER_PORT     equ 0x61
213     PIT_CTL           equ 0x43
214     PIT_PROG          equ 0xb6      ; 0b10110110: 10 (chan 2) 11 (read LSB/MSB) 011 (mode 3)
215     PIT_CHAN2         equ 0x42
216     PIT_FREQ          equ 0x1234DD
217
218     playMusic:
219         ; musicPos contains two bytes.
220         ; The first, which will become al, tells us the note we're on. e.g. the 43rd note
221         ; The second, which will become ah, tells us the position within the note. e.g. 31
222         ; clock cycles left
223         ; A clock cycle is 1/18.2 s
224         mov     ax, [musicPos]
225         dec     ah ; the position within the note
226
227         jz      .nextNote ; if we're at the last position within a note
228
229         ; if not, store back to memory, and return
230         mov     [musicPos], ax
231
232         ; we want to put a space at the end of a note just before we switch to the next note
233         ; If we're on the very last cycle of a note, blank out the sound
234         cmp     ah, 1
235         je      .space
236
237         ret
238
239     .space:
240         ; this magic incantation tells the speaker to be quiet.
241         mov     al, [portval]
242         out     SPEAKER_PORT, al
243
244         ret
245
246     .nextNote:
247         inc     al ; go to the next note
248
249         cmp     al, NOTE_NUM ; see if we've reached the end of the music
250         jne     .after
251         mov     al, 0 ; if so, go back to the beginning
252
253     .after:
254         ; figure out where the next note is in memory
255         mov     bl, al
256         mov     bh, 0
257         shl     bx, 2
258
259         ; pull in the number of cycles that we specified for the next note
260         mov     ah, [musicData + bx]
261         ; multiply by 4 to slow down the tempo
262         ; If you remove this, it's really fun to listen to . . .
263         shl     ah, 2
264
265         ; I think we're all done with messing around with ax, so we can go ahead and store
266         ; it back in it's place
267         mov     [musicPos], ax
268
269         ; get the current frequency to play
270         mov     bx, [musicData + bx + 2]

```

```

270     ; now we need to play that music
271     ; copied/edited from example code
272
273     ; Capture initial speaker state
274     in     al, SPEAKER_PORT
275     and    al, 0xfc
276     mov    [portval], al
277
278     ; Program PIT channel 2 to count at (0x1234DD / freq) [to generate that frequency]
279     ; NASM has already done the math below, since DOS-BOX doesn't support the divide
    instructions
280
281     mov    al, PIT_PROG
282     out    PIT_CTL, al
283     mov    al, bl
284     out    PIT_CHAN2, al
285     mov    al, bh
286     out    PIT_CHAN2, al
287
288     ; Turn on the speaker
289     mov    al, [portval]
290     or     al, 3
291     out    SPEAKER_PORT, al
292
293     ret
294
295 ; end really gross code
296 ;
-----
297
298 ;
-----
299 ; Start Mandelbrot code
300 ;Reference: http://jonisalonen.com/2013/lets-draw-the-mandelbrot-set/
301
302
303 ;Using half of the columns to only draw to half the screen
304 PpR equ 160 ; 160 pixels per row/scanline
305 RpS equ 200 ; 200 rows per screen/framebuffer
306 ITERATIONS equ 256
307 mandelbrot_task:
308
309     ; Set up ES to be our framebuffer segment
310     ;Needs to be done since es can be a lot of things in this program
311     mov    ax, 0xA000
312     mov    es, ax
313
314 ;Taken from example file mouspal in the class directory
315 .palcycle:
316     ; Select starting color (DI) for VGA palette transformation
317     ; (Color transforms "wrap" around, so if we start at color
318     ; 200, the first 56 colors in the table will go in palette
319     ; slots 200-255, then the remaining 200 will go in 0-199...)
320     mov    dx, 0x3C8 ; "starting color" port
321     mov    ax, di
322     xor    ah, ah
323     out    dx, al
324
325     ; Blast the color table out to the VGA registers
326     mov    cx, 256
327     mov    dx, 0x3C9 ; "R/G/B data" port
328     mov    si, palette ; source = palette array
329 .palloop:
330     lodsb
331     out    dx, al ; Red
332     lodsb
333     out    dx, al ; Green

```

```

334     lodsb
335     out dx, al      ; Blue
336     loop     .palloop
337
338
339     ; Clear screen to black (copy 320*200 byte of ZERO to the framebuffer)
340     mov al, 0
341     mov cx, 320*200
342     mov di, 0
343     rep stosb
344
345
346 ;Begin Mandelbrot logic
347     mov cx, 0
348 ;Row Loop
349 .compare_row:
350     cmp cx, RpS
351     jge .end_comp_row
352     ;Save cx for after inner loop
353     push cx
354     mov cx, 0
355
356 ;Column Loop
357 .compare_col:
358     cmp cx, PpR
359     jge .end_comp_col
360
361     ;Get current row value into bx and save again
362     pop bx
363     push bx
364
365     ;So here we do a lot of flops to do a little bit of math
366     ;The formula is essentially this,
367     ;while(x^2 + y^2 < 4 and iterations < MAX_ITER):
368     ;We actually do complex math with doing complex math by treating a complex number
369     ;as an (x,y) pair
370     ;Heres what we do
371     ;x_new = x*x - y*y + c_re;
372     ;y = 2*x*y + c_im;
373     ;x = x_new;
374     ;increment iterations
375     ;loop
376     mov word[iteration], 0
377     fld dword[zero]
378     fst dword[x]
379     fst dword[x0]
380     fst dword[y0]
381     fstp dword[y]
382
383     mov [temp_col], cx
384     fild word [temp_col]
385     fsub dword [width_adj]
386     fmul dword [const_four]
387     fdiv dword [width]
388     fstp dword [x0]
389
390     mov [temp_row], bx
391     fild word [temp_row]
392     fsub dword [height_adj]
393     fmul dword [const_four]
394     fdiv dword [width]
395     fstp dword [y0]
396
397 .float_comp:
398     fld dword[x]
399     fmul dword[x]
400     fstp dword[x2]
401     fld dword[y]
402     fmul dword[y]

```

```

402     fstp dword[y2]
403     fld dword[x2]
404     fld dword[y2]
405     faddp
406     fld dword[const_four]
407
408     ;Because 8086 floating point operations dont' support direct x87 flag compares we
409     do a little bit of magic
410     ;Fcmop will set the flags on the x87
411     fcomp
412     ;We grab the x87 status word
413     fnstsw word [status_word]
414     mov ax, [status_word]
415     ;We need to get the values of three specifci bits of the status word namely the 9th,
416     11th, and 15th bits
417     ;Note 17664 == 1000101000000000
418     mov di, 17664
419     and ax, di
420     ;If we get 0 back we are still less than 4
421     cmp ax, 0
422
423     je .loopy
424     ;Here we check if we are equeal to 4
425     mov ax, [status_word]
426     and ax, 16384
427     cmp ax, 16384
428     jne .end_crazy_pls
429
430 .loopy:
431     ;Check our iterations to see if we are still valid
432     cmp word [iteration], ITERATIONS
433     jge .end_crazy_pls
434
435     ;Calculate new x and y values
436     fld dword[x2]
437     fld dword[y2]
438     fsub
439     fld dword[x0]
440     fadd
441     fst dword[new_x]
442
443     fld dword[const_two]
444     fld dword[x]
445     fmul
446     fld dword[y]
447     fmul
448     fld dword[y0]
449     fadd
450     fst dword[y]
451     fld dword[new_x]
452     fst dword[x]
453     inc word[iteration]
454
455     jmp .float_comp
456
457 .end_crazy_pls:
458     ;Check iterations to see if we are still bounded
459     cmp word[iteration], ITERATIONS
460     jge .end_of_all
461     ;If bounded, get the current position on the screen
462     ;320*row + col
463     mov ax, 320
464     imul bx
465     push ax
466     mov ax, cx
467     pop dx
468     add ax, dx
469
470     push bx
471     mov bx, ax

```

```

469     mov ax, [iteration]
470     add ax, [_cur_pal_offset]
471     ;Set color to number of iterations + palette_offset
472     mov byte[es:bx], al
473     pop bx
474     jmp .end_of_all
475 .end_of_all:
476     inc cx
477     jmp .compare_col
478 .end_comp_col:
479
480
481     pop cx
482     inc cx
483     jmp .compare_row
484 .end_comp_row:
485
486     jmp mandlebrot_task
487
488
489 section .data
490 x0 dd 0.0
491 y0 dd 0.0
492 x dd 0.0
493 new_x dd 0.0
494 y dd 0.0
495 x2 dd 0.0
496 y2 dd 0.0
497 zero dd 0.0
498 width_adj dd 80.0
499 width dd 160.0
500 height_adj dd 100.0
501 height dd 200.0
502 const_four dd 4.0
503 const_two dd 2.0
504 temp_col dw 0
505 temp_row dw 0
506 iteration dw 0
507 junk dq 0.0
508 status_word dw 0
509 temp dw 0
510
511 ;Stolen from example file mouspal.asm
512 ; Smooth-blending 256 color palette
513 ; generated by a Python script
514 ; (RGB values in the range 0-63)
515 palette db 0, 0, 0
516         db 1, 0, 0
517         db 2, 0, 0
518         db 3, 0, 0
519         db 4, 0, 0
520         db 5, 0, 0
521         db 6, 0, 0
522         db 7, 0, 0
523         db 8, 0, 0
524         db 9, 0, 0
525         db 10, 0, 0
526         db 11, 0, 0
527         db 12, 0, 0
528         db 13, 0, 0
529         db 14, 0, 0
530         db 15, 0, 0
531         db 16, 0, 0
532         db 17, 0, 0
533         db 18, 0, 0
534         db 19, 0, 0
535         db 20, 0, 0
536         db 21, 0, 0
537         db 22, 0, 0

```


538	db	23	,	0	,	0
539	db	24	,	0	,	0
540	db	25	,	0	,	0
541	db	26	,	0	,	0
542	db	27	,	0	,	0
543	db	28	,	0	,	0
544	db	29	,	0	,	0
545	db	30	,	0	,	0
546	db	31	,	0	,	0
547	db	32	,	0	,	0
548	db	33	,	0	,	0
549	db	34	,	0	,	0
550	db	35	,	0	,	0
551	db	36	,	0	,	0
552	db	37	,	0	,	0
553	db	38	,	0	,	0
554	db	39	,	0	,	0
555	db	40	,	0	,	0
556	db	41	,	0	,	0
557	db	42	,	0	,	0
558	db	43	,	0	,	0
559	db	44	,	0	,	0
560	db	45	,	0	,	0
561	db	46	,	0	,	0
562	db	47	,	0	,	0
563	db	48	,	0	,	0
564	db	49	,	0	,	0
565	db	50	,	0	,	0
566	db	51	,	0	,	0
567	db	52	,	0	,	0
568	db	53	,	0	,	0
569	db	54	,	0	,	0
570	db	55	,	0	,	0
571	db	56	,	0	,	0
572	db	57	,	0	,	0
573	db	58	,	0	,	0
574	db	59	,	0	,	0
575	db	60	,	0	,	0
576	db	61	,	0	,	0
577	db	62	,	0	,	0
578	db	63	,	0	,	0
579	db	63	,	0	,	0
580	db	63	,	1	,	0
581	db	63	,	2	,	0
582	db	63	,	3	,	0
583	db	63	,	4	,	0
584	db	63	,	5	,	0
585	db	63	,	6	,	0
586	db	63	,	7	,	0
587	db	63	,	8	,	0
588	db	63	,	9	,	0
589	db	63	,	10	,	0
590	db	63	,	11	,	0
591	db	63	,	12	,	0
592	db	63	,	13	,	0
593	db	63	,	14	,	0
594	db	63	,	15	,	0
595	db	63	,	16	,	0
596	db	63	,	17	,	0
597	db	63	,	18	,	0
598	db	63	,	19	,	0
599	db	63	,	20	,	0
600	db	63	,	21	,	0
601	db	63	,	22	,	0
602	db	63	,	23	,	0
603	db	63	,	24	,	0
604	db	63	,	25	,	0
605	db	63	,	26	,	0
606	db	63	,	27	,	0

607	db	63	28	0
608	db	63	29	0
609	db	63	30	0
610	db	63	31	0
611	db	63	32	0
612	db	63	33	0
613	db	63	34	0
614	db	63	35	0
615	db	63	36	0
616	db	63	37	0
617	db	63	38	0
618	db	63	39	0
619	db	63	40	0
620	db	63	41	0
621	db	63	42	0
622	db	63	43	0
623	db	63	44	0
624	db	63	45	0
625	db	63	46	0
626	db	63	47	0
627	db	63	48	0
628	db	63	49	0
629	db	63	50	0
630	db	63	51	0
631	db	63	52	0
632	db	63	53	0
633	db	63	54	0
634	db	63	55	0
635	db	63	56	0
636	db	63	57	0
637	db	63	58	0
638	db	63	59	0
639	db	63	60	0
640	db	63	61	0
641	db	63	62	0
642	db	63	63	0
643	db	63	63	0
644	db	63	63	1
645	db	63	63	2
646	db	63	63	3
647	db	63	63	4
648	db	63	63	5
649	db	63	63	6
650	db	63	63	7
651	db	63	63	8
652	db	63	63	9
653	db	63	63	10
654	db	63	63	11
655	db	63	63	12
656	db	63	63	13
657	db	63	63	14
658	db	63	63	15
659	db	63	63	16
660	db	63	63	17
661	db	63	63	18
662	db	63	63	19
663	db	63	63	20
664	db	63	63	21
665	db	63	63	22
666	db	63	63	23
667	db	63	63	24
668	db	63	63	25
669	db	63	63	26
670	db	63	63	27
671	db	63	63	28
672	db	63	63	29
673	db	63	63	30
674	db	63	63	31
675	db	63	63	32

676	db	63	63	33
677	db	63	63	34
678	db	63	63	35
679	db	63	63	36
680	db	63	63	37
681	db	63	63	38
682	db	63	63	39
683	db	63	63	40
684	db	63	63	41
685	db	63	63	42
686	db	63	63	43
687	db	63	63	44
688	db	63	63	45
689	db	63	63	46
690	db	63	63	47
691	db	63	63	48
692	db	63	63	49
693	db	63	63	50
694	db	63	63	51
695	db	63	63	52
696	db	63	63	53
697	db	63	63	54
698	db	63	63	55
699	db	63	63	56
700	db	63	63	57
701	db	63	63	58
702	db	63	63	59
703	db	63	63	60
704	db	63	63	61
705	db	63	63	62
706	db	63	63	63
707	db	63	63	63
708	db	63	63	63
709	db	62	62	62
710	db	61	61	61
711	db	60	60	60
712	db	59	59	59
713	db	58	58	58
714	db	57	57	57
715	db	56	56	56
716	db	55	55	55
717	db	54	54	54
718	db	53	53	53
719	db	52	52	52
720	db	51	51	51
721	db	50	50	50
722	db	49	49	49
723	db	48	48	48
724	db	47	47	47
725	db	46	46	46
726	db	45	45	45
727	db	44	44	44
728	db	43	43	43
729	db	42	42	42
730	db	41	41	41
731	db	40	40	40
732	db	39	39	39
733	db	38	38	38
734	db	37	37	37
735	db	36	36	36
736	db	35	35	35
737	db	34	34	34
738	db	33	33	33
739	db	32	32	32
740	db	31	31	31
741	db	30	30	30
742	db	29	29	29
743	db	28	28	28
744	db	27	27	27

```

745     db 26, 26, 26
746     db 25, 25, 25
747     db 24, 24, 24
748     db 23, 23, 23
749     db 22, 22, 22
750     db 21, 21, 21
751     db 20, 20, 20
752     db 19, 19, 19
753     db 18, 18, 18
754     db 17, 17, 17
755     db 16, 16, 16
756     db 15, 15, 15
757     db 14, 14, 14
758     db 13, 13, 13
759     db 12, 12, 12
760     db 11, 11, 11
761     db 10, 10, 10
762     db 9, 9, 9
763     db 8, 8, 8
764     db 7, 7, 7
765     db 6, 6, 6
766     db 5, 5, 5
767     db 4, 4, 4
768     db 3, 3, 3
769     db 2, 2, 2
770     db 1, 1, 1
771
772
773 ;End Mandlebrot Code
774 ;-----
775
776
777 section .data
778 ; seed for random number generation
779 seed      dw 0
780
781 saved_sp   dw 0
782
783 ;number of times to run before exiting
784 timesToRun dw 10
785
786
787 ivt8_offset dw 0
788 ivt8_segment dw 0
789
790 int_msg     db "Int", 13, 10, 0
791
792 msg1        db "I am task A!", 13, 10, 0
793 msg2        db "I am task B!", 13, 10, 0
794 msg3        db "I am task C!", 13, 10, 0
795
796 ;Task stacks
797 stack1      times 256 db 0
798 stack2      times 256 db 0
799 stack3      times 256 db 0
800 stack4      times 256 db 0
801
802 stacks times 4 dw 0 ;Stack pointer array array
803
804 stack_idx dw 2 ;Stating task and current task, Zero-based
805
806 NOTE_NUM equ 130
807
808 ; first number is the number of which note in the song we're on. The second is the
; position within that note
809 musicPos db (NOTE_NUM - 1), 1
810
811 ; there are 40 notes + 2 for the amen and one for a blank space to let us regain our

```

```

sanity before it starts again
812 ; we'll make NASM do the math for us on what frequencies to use
813 ; University Hymn
814 musicData dw 4, (PIT_FREQ / 392), 2, (PIT_FREQ / 349), 2, (PIT_FREQ / 349), 4,
(PIT_FREQ / 311), 4, (PIT_FREQ / 392), 2, (PIT_FREQ / 466), 2, (PIT_FREQ / 523), 2,
(PIT_FREQ / 466), 2, (PIT_FREQ / 415), 8, (PIT_FREQ / 392), 4, (PIT_FREQ / 392), 3,
(PIT_FREQ / 440), 1, (PIT_FREQ / 440), 4, (PIT_FREQ / 466), 4, (PIT_FREQ / 523), 2,
(PIT_FREQ / 587), 2, (PIT_FREQ / 523), 2, (PIT_FREQ / 466), 2, (PIT_FREQ / 440), 8,
(PIT_FREQ / 466), 4, (PIT_FREQ / 466), 2, (PIT_FREQ / 415), 2, (PIT_FREQ / 392), 4,
(PIT_FREQ / 349), 4, (PIT_FREQ / 392), 2, (PIT_FREQ / 311), 2, (PIT_FREQ / 311), 2,
(PIT_FREQ / 349), 2, (PIT_FREQ / 349), 8, (PIT_FREQ / 392), 4, (PIT_FREQ / 415), 2,
(PIT_FREQ / 466), 2, (PIT_FREQ / 523), 4, (PIT_FREQ / 622), 4, (PIT_FREQ / 415), 2,
(PIT_FREQ / 392), 2, (PIT_FREQ / 349), 2, (PIT_FREQ / 311), 2, (PIT_FREQ / 294), 8,
(PIT_FREQ / 311), 8, (PIT_FREQ / 311), 8, (PIT_FREQ / 311), 16, 1, \
815 2, (PIT_FREQ / 415), 1, (PIT_FREQ / 392), 1, (PIT_FREQ / 415), 1,
(PIT_FREQ / 349), 2, (PIT_FREQ / 415), 1, (PIT_FREQ / 466), 1, (PIT_FREQ /
494), 1, (PIT_FREQ / 523), 1, (PIT_FREQ / 554), 1, (PIT_FREQ / 587), 2,
(PIT_FREQ / 622), 2, (PIT_FREQ / 622), 2, (PIT_FREQ / 622), 1, (PIT_FREQ /
554), 1, (PIT_FREQ / 523), 2, (PIT_FREQ / 523), 1, (PIT_FREQ / 494), 1,
(PIT_FREQ / 523), 6, (PIT_FREQ / 523), 1, (PIT_FREQ / 494), 1, (PIT_FREQ /
523), 2, (PIT_FREQ / 523), 1, (PIT_FREQ / 494), 1, (PIT_FREQ / 523), 2,
(PIT_FREQ / 622), 1, (PIT_FREQ / 523), 1, (PIT_FREQ / 622), 4, (PIT_FREQ /
554), 3, (PIT_FREQ / 466), 1, (PIT_FREQ / 466), 2, (PIT_FREQ / 466), 1,
(PIT_FREQ / 440), 1, (PIT_FREQ / 466), 2, (PIT_FREQ / 466), 1, (PIT_FREQ /
440), 1, (PIT_FREQ / 466), 6, (PIT_FREQ / 554), 1, (PIT_FREQ / 523), 1,
(PIT_FREQ / 466), 1, (PIT_FREQ / 523), 2, (PIT_FREQ / 622), 1, (PIT_FREQ /
622), 2, (PIT_FREQ / 699), 2, (PIT_FREQ / 699), 6, (PIT_FREQ / 466), 2,
(PIT_FREQ / 622), 2, (PIT_FREQ / 622), 1, (PIT_FREQ / 554), 1, (PIT_FREQ /
523), 2, (PIT_FREQ / 523), 1, (PIT_FREQ / 494), 1, (PIT_FREQ / 523), 6,
(PIT_FREQ / 523), 1, (PIT_FREQ / 494), 1, (PIT_FREQ / 523), 2, (PIT_FREQ /
523), 1, (PIT_FREQ / 494), 1, (PIT_FREQ / 523), 1, (PIT_FREQ / 554), 1,
(PIT_FREQ / 523), 1, (PIT_FREQ / 466), 1, (PIT_FREQ / 392), 4, (PIT_FREQ /
466), 3, (PIT_FREQ / 415), 1, (PIT_FREQ / 415), 2, (PIT_FREQ / 415), 1,
(PIT_FREQ / 392), 1, (PIT_FREQ / 415), 2, (PIT_FREQ / 494), 1, (PIT_FREQ /
466), 1, (PIT_FREQ / 415), 5, (PIT_FREQ / 831), 1, (PIT_FREQ / 415), 1,
(PIT_FREQ / 466), 1, (PIT_FREQ / 523), 1, (PIT_FREQ / 622), 1, (PIT_FREQ /
415), 1, (PIT_FREQ / 466), 1, (PIT_FREQ / 523), 1, (PIT_FREQ / 622), 1,
(PIT_FREQ / 311), 1, (PIT_FREQ / 349), 1, (PIT_FREQ / 523), 4, (PIT_FREQ /
466), 2, (PIT_FREQ / 415), 1, (PIT_FREQ / 415), 16, 1
816 ; other music . . .
817 portval dw 0
818
819 ;External Variables used in C functions
820 global _cur_pal_offset
821 _cur_pal_offset dw 0
822 global _currPos0
823 _currPos0 dw 0
824 global _currPos1
825 _currPos1 dw 60
826 global _currPos2
827 _currPos2 dw 120

```