Mercurial > hg > index.cgi

view dod.s @ 53:bb39e4af7e21 tip

- log
- graph
- tags
- bookmarks
- branches

- changeset
- browse

- **file**
- latest
- diff
- comparison
- annotate
- file log
- raw

- help

```
Fix reference to NMI vector

The initializer sets the NMI vector, not the FIRQ vector. It still makes no
sense, but there you go.
```

author William Astle <lost@l-w.ca>
date Sun, 28 Dec 2014 10:48:13 -0700 (2014-12-28)
parents 47ffee6789b1
children

line source **line wrap: on**

```
; Dungeons of Daggorath
;
; Original game copyright Â© 1982 by Dyna Micro
;
; The code contained in this file is not the original source code to Dungeons of Daggorath. It was
; constructed by William Astle in 2013 by disassembling the Dungeons of Daggorath ROM.
;
; According to a web page retrieved from http://frodpod.tripod.com/lisence.html on May 24, 2013,
; this endeavour is permitted. In case the web page becomes unavailable, and because it contains what
; I believe to be important credit information, I have reproduced the text of it below:
;
;*******************************************************************************************************
;* Grant of license to reproduce Dungeons of Daggorath
;*
;* My name is Douglas J. Morgan.  I was the president of DynaMicro, Inc. (since dissolved), the company
;* which conceived, created and wrote Dungeons of Daggorath, a best selling Radio Shack Color Computer
;* adventure game.
;*
;* I have examined the contract I signed with Radio Shack for their license of the game.  The contract
;* provides that Radio Shack shall have an exclusive license to manufacture and produce the game, but
;* that said exclusive license shall revert to a non-exclusive license should Radio Shack cease to
;* produce and sell the game.  To the best of my knowledge, they have not produced the game for many
;* years.  Thus, it is my belief that the right to grant a license for the game has reverted to me.
;*
;* I hereby grant a non-exclusive permanent world-wide license to any and all Color Computer site
;* administrators, emulator developers, programmers or any other person or persons who wish to develop,
;* produce, duplicate, emulate, or distribute the game on the sole condition that they exercise every
;* effort to preserve the game insofar as possible in its original and unaltered form.
;*
;* The game was a labor of love.  Additional credits to Phillip C. Landmeier - who was my partner and
;* who originally conceived the vision of the game and was responsible for the (then) state of the art
;* sounds and realism, to April Landmeier, his wife - the artist who drew all the creatures as well as
;* all the artwork for the manual and game cover, and to Keith Kiyohara - a gifted programmer who helped
;* write the original game and then contributed greatly to compressing a 16K game into 8K so that it
;* could be carried and produced by Radio Shack.
;*
;* The game did very well for us.  I give it to the world with thanks to all who bought it, played it
;* or enjoyed it.
;*
;* There is one existing copy of the original source code.  Anyone willing to pay for the copying of
;* the listing (at Kinko's) and shipment to them, who intends to use it to enhance or improve the emulator
;* versions of the game is welcome to it.
;*
;* Verification of this license grant or requests for the listing can be made by contacting Louis Jordan,
;* Thank you.
;*************************************************************************************************************
;
; Louis Jordan's email address is given as louisgjordan@yahoo.com in a hyperlink in the above statement.
;
; It is my belief that this endeavor to disassemble Dungeons of Daggorath is in compliance with the above
; license grant. I have done so for my own amusement and for the challenge. I have also done so because
; I failed to elicit a response from Louis Jordan as described in the license grant. I am not surprised
; that I received no reply given that the page above was put online during or prior to 2006.

; some utility macros
dod             macro noexpand
                swi
                fcb \1
                ifeq \1-$1B
                fcb \2
                endc
                endm

skip2           macro noexpand
                fcb $8c
                endm


; macros for color basic ROM calls
romcall         macro noexpand
                swi2
                fcb \1
                endm
```

```

; set lighting for render
setlighting     macro noexpand
                dod S00
                endm
; draw a graphic with graphic data at (X)
drawgraphic     macro noexpand
                dod S01
                endm
; render a packed string (immediate data)
renderstrimmp   macro noexpand
                dod S02
                endm
; render a packed string from (X)
renderstr       macro noexpand
                dod S03
                endm
; render character in A
renderchar      macro noexpand
                dod S04
                endm
; decode a 5 bit packed string from (X) to stringbuf
decodestrsb     macro noexpand
                dod S05
                endm
; decode a 5 bit packed string from (X) to (U)
decodestr       macro noexpand
                dod S06

                endm
; generate an 8 bit random number in A
getrandom       macro noexpand
                dod S07
                endm
; clear graphics screen currently visible; return parameter pointer in U
cleargfx1       macro noexpand
                dod S08
                endm
; clear graphics screen currently used for drawing; return parameter pointer in U
cleargfx2       macro noexpand
                dod S09
                endm
; clear the status line
clearstatus     macro noexpand
                dod S0A
                endm
; clear the command entry area
```

```
clearcommand    macro noexpand
                dod S0B
                endm
; check for death, fainting, or recovery, and calculate how long before next
; damage reduction tick
checkdamage     macro noexpand
                dod S0C
                endm
; update the inventory on the status line
updatestatus    macro noexpand
                dod S0D
                endm
; update dungeon display
updatedungeon   macro noexpand
                dod S0E
                endm
; do a newline, show prompt, and cursor
showprompt      macro noexpand
                dod S0F
                endm

; do a delay for about 1.33 seconds
delay           macro noexpand
                dod S10
                endm

; set a block of memory (from X to U-1) to $00
clearblock      macro noexpand
                dod S11
                endm
; set a block of memory (from X to U-1) to $ff
setblock        macro noexpand
                dod S12
                endm
; fade in the image at (X) with sound effects at scale 1.0, clear status and command area
fadeinclrst     macro noexpand
                dod S13
                endm
; fade in the image at (X) with sound effects at scale 1.0, clear command area
fadein          macro noexpand
                dod S14
                endm
; fade out image at (X) with sound effects, clear command area
fadeout         macro noexpand
                dod S15
                endm
```

```
; display the PREPARE! screen
showprepare     macro noexpand
                dod S16
                endm
; create object of type in A
createobject    macro noexpand
                dod S17
                endm
; set object specs (object pointer in U)
setobjectspecs  macro noexpand
                dod S18
                endm
; reset display and show dungeon
resetdisplay    macro noexpand
                dod S19
                endm
; generate a level
createlevel     macro noexpand
                dod S1A
                endm
; play a sound number from immediate data at full volume
playsoundimm    macro noexpand
                dod S1B,\1
                endm
; play sound specified in A, volume in B
playsound       macro noexpand
                dod S1C
                endm

; ROM call numbers
POLCAT          equ 0
CSRDON          equ 4
BLKIN           equ 6
BLKOUT          equ 8
WRTLDR          equ 12

ROMTAB          equ $A000

BLKTYP          equ $7c
BLKLEN          equ $7d
CBUFAD          equ $7e

RESVEC          equ $A027

; SWI routines
S00             equ 0
```

```
S01             equ 1
S02             equ 2
S03             equ 3
S04             equ 4
S05             equ 5
S06             equ 6
S07             equ 7
S08             equ 8
S09             equ 9
S0A             equ $0A
S0B             equ $0B
S0C             equ $0C
S0D             equ $0D
S0E             equ $0E
S0F             equ $0F
S10             equ $10
S11             equ $11
S12             equ $12
S13             equ $13
S14             equ $14
S15             equ $15
S16             equ $16
S17             equ $17
S18             equ $18
S19             equ $19
S1A             equ $1A
S1B             equ $1B
S1C             equ $1C

PIA0            equ $ff00
PIA1            equ $ff20
SAMREG          equ $ffc0
TOPRAM          equ $4000
STACK           equ $1000

; the direct page
                org $200
zero            rmb 2                           ; initialized to $0000
V202            rmb 1                           ; apparently unused
allones         rmb 2                           ; initialized to $ffff
horizcent       rmb 2                           ; center coordinate for scaled graphics (X)
vertcent        rmb 2                           ; center coordinate for scaled graphics (Y)
screenvis       rmb 2                           ; pointer to the parameter block of the currently shown screen
screendraw      rmb 2                           ; pointer to the parameter block of the screen to use for
drawing
demoseqptr      rmb 2                           ; pointer to demo game command sequence
```

```
objectfree      rmb 2                           ; pointer to next free object data slot
linebuffptr     rmb 2                           ; line input buffer pointer
playerloc       rmb 2                           ; current player position in maze
carryweight     rmb 2                           ; how much weight the player is currently carrying (for movement cost)
; powerlevel, magicoff, magicdef, physoff, physdef, and damagelevel must remain in the same specific order
; with the same spacing between them in order to match the same structure used by the creature
; data.
powerlevel      rmb 2                           ; player power
magicoff        rmb 1                           ; magical attack value (player)
magicdef        rmb 1                           ; magical defense value (player)
physoff         rmb 1                           ; physical attack value (player)
physdef         rmb 1                           ; physical defense value (player)
lefthand        rmb 2                           ; pointer to object carried in left hand
righthand       rmb 2                           ; pointer to object carried in right hand
damagelevel     rmb 2                           ; player damage level
facing          rmb 1                           ; the direction the player is facing
curtorch        rmb 2                           ; pointer to currently mounted torch
baselight       rmb 2                           ; base light level in dungeon
nokeyboard      rmb 1                           ; set if no keyboard operations should be done during IRQ
backpack        rmb 2                           ; pointer to first item in backpack
creaturefreeze  rmb 1                           ; nonzero means creatures are frozen
levbgmask       rmb 1                           ; the current level background colour mask
lightlevel      rmb 1                           ; the current light level, $ff means dark
lightcount      rmb 1                           ; counter between pixels when drawing lines
ybeg            rmb 2                           ; start Y coord for line drawing
xbeg            rmb 2                           ; start X coord for line drawing
yend            rmb 2                           ; end Y coord for line drawing
xend            rmb 2                           ; end X coord for line drawing
xcur            rmb 3                           ; current X coordinate when drawing line
ycur            rmb 3                           ; current Y coordinate when drawing line
xpstep          rmb 3                           ; difference in X coordinate between pixels when drawing line
ypstep          rmb 3                           ; difference in Y coordinate between pixels when drawing line
pixelcount      rmb 2                           ; number of pixels to draw in a line
xbstep          rmb 1                           ; the offset to add to pointer when moving to new byte (line drawing)
xystep          rmb 1                           ; the offset to add to pointer when moving to new row (line drawing)
drawstart       rmb 2                           ; start address of drawing area (line drawing)
drawend         rmb 2                           ; end address of drawing area (line drawing)
                rmb 4                           ; *unused*
horizscale      rmb 1                           ; horizontal scaling factor for rendering
vertscale       rmb 1                           ; vertical scaling factor for rendering
polyfirst       rmb 1                           ; for rendering images - set if this is the first vertex
lastunscalex    rmb 2                           ; most recent unscaled X coordinate for rendering
lastunscaley    rmb 2                           ; most recent unscaled Y coordinate for rendering
```

```
soundseqseed    rmb 2                                   ; sound: sequence generator seed
                rmb 1                                   ; *unused*
sndtemp         rmb 1                                   ; sound: temporary storage for dac value
                rmb 1                                   ; *unused*
sndampmult      rmb 2                                   ; sound: amplitude multiplier for volume slides (MSB is used)
sndampstep      rmb 2                                   ; sound: amplitude step for volume slides
soundrepeat     rmb 1                                   ; sound: repeat counter
                rmb 1                                   ; *unused*
soundvol        rmb 1                                   ; sound: volume multiplier for sound playing
soundrepeat2    rmb 1                                   ; sound: repeat counter for scorpion, wraith, and viper sounds
sndlowtonedel   rmb 2                                   ; sound: low tone wave delay for dual tone generator
sndhitonedel    rmb 2                                   ; sound: high tone wave delay for dual tone generator
                rmb 4                                   ; *unused*
randomseed      rmb 3                                   ; random number generator seed value
effectivelight  rmb 1                                   ; effective light level in dungeon
effectivemlight rmb 1                                   ; effective magical light level in dungeon (also used for alternative light schemes)
savedefflight   rmb 1                                   ; effective light level saved during fainting
                rmb 2                                   ; *unused*
movehalf        rmb 1                                   ; set if rendering half step for MOVE
movebackhalf    rmb 1                                   ; set if rendering half step for MOVE BACK
rendermagic     rmb 1                                   ; set if rendering should use magical illumination
                rmb 1                                   ; *unused*
waitnewgame     rmb 1                                   ; set if waiting for keypress to start new game
kwmatch         rmb 1                                   ; for tracking if a match is found when looking up a keyword
kwcount         rmb 1                                   ; counter during keyword list lookup
                rmb 1                                   ; *unused*
kwexact         rmb 1                                   ; set if keyword lookup matched exactly

temploc         rmb 2                                   ; working coordinates during various processing steps
genpathlen      rmb 1                                   ; number of steps to dig during maze generation
                rmb 2                                   ; *unused*
currentlevel    rmb 1                                   ; currently playing dungeon level
creaturecntptr  rmb 2                                   ; pointer to creature count table for the current level
                rmb 2                                   ; *unused*
holetabptr      rmb 2                                   ; pointer to the hole table for this level
gencurcoord     rmb 2                                   ; current coordinates when generating maze
curdir          rmb 1                                   ; current direction we're processing
renderdist      rmb 1                                   ; distance from "camera" for rendering
objectcount     rmb 1                                   ; number of objects of specific type to create when initializing
objectlevel     rmb 1                                   ; starting level (minimum) of object being created
parseobjtype    rmb 1                                   ; the type of object parsed from command
parseobjtypegen rmb 1                                   ; the generic type of the object parsed
parsegenobj     rmb 1                                   ; nonzero if a generic object was parsed
objiterstart    rmb 1                                   ; for iterating over object list, zero means start, nonzero means underway
```

```
objiterptr      rmb 2                                   ; current pointer during object list iteration
showseer        rmb 1                                   ; nonzero means to show creatures when displaying a scroll
clockctrs       rmb 1                                   ; 60Hz tick (triggers 10 times per second)
                rmb 1                                   ; 10Hz tick (triggers once per second)
                rmb 1                                   ; 1Hz tick (triggers once per minute)
                rmb 1                                   ; 1 minute tick (triggers once per hour)
                rmb 1                                   ; 1 hour tick (triggers once per day)
                rmb 1                                   ; 1 day tick (overflows once every 256 days)
disablesched    rmb 1                                   ; set to nonzero to disable timer handling
dofadesound     rmb 1                                   ; nonzero if we're doing the fade sound effect thing
fadesoundval    rmb 1                                   ; the DAC value to use for the fade sound (complemented every tick)
enablefadesound rmb 1                                   ; nonzero means fading will be done with the sound effect
schedlists      rmb 2                                   ; notional "not active" queue?
                rmb 2                                   ; 60Hz tick event list
                rmb 2                                   ; 10Hz tick event list
                rmb 2                                   ; 1Hz tick event list
                rmb 2                                   ; 1 minute tick event list
                rmb 2                                   ; 1 hour tick event list
                rmb 2                                   ; the "ready" list (also 1 day tick event list)
hidestatus      rmb 1                                   ; nonzero will cause the command processor to clear and reset on input (status line hidden)
heartctr        rmb 1                                   ; number of ticks until next heart beat
heartticks      rmb 1                                   ; number of ticks between heart beat
heartstate      rmb 1                                   ; zero = contracted heart, ff = expanded heart
enableheart     rmb 1                                   ; nonzero means heartbeat is running
displayptr      rmb 2                                   ; pointer to routine to display the main dungeon area
pageswap        rmb 1                                   ; nonzero means we're ready to swap graphics screens during IRQ
dungeonchg      rmb 1                                   ; nonzero if the dungeon display should be updated
columnctr       rmb 1                                   ; column counter/tracker for displaying inventory list
textother       rmb 1                                   ; nonzero means nonstandard text location
loadsaveflag    rmb 1                                   ; load/save flag - <0 = ZLOAD, >0 = ZSAVE, 0 = regular init
schedtabfree    rmb 2                                   ; pointer to next free entry in the scheduling table
readylistchg    rmb 1                                   ; nonzero if the ready list processing should be restarted
keybufread      rmb 1                                   ; keyboard buffer read offset
keybufwrite     rmb 1                                   ; keyboard buffer write offset
                rmb 3                                   ; *unused*
accum0          rmb 3                                   ; temporary 24 bit accumulator
accum1          rmb 3                                   ; temporary 24 bit accumulator
accum2          rmb 1                                   ; temporary 8 bit accumulator
                rmb 9                                   ; *unused*
keybuf          rmb 32                                  ; keyboard ring buffer
linebuff        rmb 32                                  ; line input buffer
linebuffend     equ *                                   ; end of line input buffer
                rmb 2                                   ; *unused*
wordbuff        rmb 32                                  ; buffer used for parsing words from the line
```

```
wordbuffend     equ *                           ; end of word buffer
                rmb 2                           ; *unused*
stringbuf       rmb 34                          ; temporary buffer used for decoding immediate packed strings
fontbuf         rmb 10                          ; temporary buffer used for decoding font data
cmddecodebuff   rmb 31                          ; buffer used for decoding commands
; These are descriptors used for controlling the rendering engine. Each one is 8 bytes long and is defined as
follows:
; 0-1   start address of text area (memory)
; 2-3   number of character cells in text area
; 4-5   current cursor position within text area
; 6     background colour mask for the text area
; 7     whether to render text on the secondary screen
;
; There are three text areas:
; infoarea      this is text rendered in the main dungeon display area
; statusarea    this is text rendered on the status line
; commandarea   this is text rendered in the command area
infoarea        rmb 2                           ; screen start address of info text area
                rmb 2                           ; number of character cells in info text area
                rmb 2                           ; current cursor position in info text area
                rmb 1                           ; background colour mask for info text area
                rmb 1                           ; nonzero if info text area should not be rendered on
secondary screen
statusarea      rmb 2                           ; screen start address of status line area
                rmb 2                           ; number of character cells in status line area
                rmb 2                           ; cursor position in status line area
                rmb 1                           ; background colour mask for status line area


                rmb 1                           ; nonzero if status line text should not be rendered on
secondary screen
commandarea     rmb 2                           ; start offset of the command entry area
                rmb 2                           ; numer of character cells in command entry area
                rmb 2                           ; current cursor position in entry area
                rmb 1                           ; background colour mask of background area
                rmb 1                           ; nonzero if main text should not be rendered on secondary
screen
creaturecounts  rmb 12                          ; creature counts for level 1
                rmb 12                          ; creature counts for level 2
                rmb 12                          ; creature counts for level 3
                rmb 12                          ; creature counts for level 4
                rmb 12                          ; creature counts for level 5
creaturetab     rmb 32*17                       ; the creatures currently active on this level
mazedata        rmb $400                        ; the actual room data for the current level
neighbourbuff   rmb 9                           ; buffer for calculating neighbors in maze generation
schedtab        rmb $10a                        ; scheduler entries
emptyhand       rmb 14                          ; "object" information for empty hand
```

```
objecttab       rmb 72*14                       ; the object data table (room for 72 entries)
datatop         equ *


                org $c000
START           ldu #dodemo                     ; point to the demo setup routine
                bra LC008                       ; go handle the game
LC005           ldu #dogame                     ; point to the real game setup routine
LC008           lds #STACK                      ; put the stack somewhere safe
                ldx #PIA0                       ; point at PIA0
                ldd #$34fa                      ; initializers for the PIA
                sta 3,x                         ; set data mode, interrupts disabled (side B)
                sta 1,x                         ; set data mode, interrupts disabled (side A)
                ldx #PIA1                       ; point at PIA1
                sta 1,x                         ; set data mode, interrupts disabled (side A)
                clr 3,x                         ; set direction mode for side B
                stb 2,x                         ; set VDG and single bit sound to output
                lda #$3c                        ; flags for data mode, no interrupts, sound enabled
                sta 3,x                         ; set side B for data mode
                ldd #$2046                      ; SAM value for "pmode 4" graphics, screen at $1000
                jsr setSAM                      ; go set the SAM register
                lda #$f8                        ; value for "pmode 4", color set 1
                sta 2,x                         ; set VDG mode
                ldx #zero                       ; point to start of variables
LC030           clr ,x+                         ; clear a byte
                cmpx #TOPRAM                    ; are we at the top of memory?
                blo LC030                       ; brif not
                stu ,--s                        ; set return address to the game initialization routine

                lda #zero/256                   ; point to MSB of direct page
                tfr a,dp                        ; set DP appropriately
                setdp zero/256                  ; tell the assembler about DP
                ldy #LD7E8                      ; point to variable initialization table
LC041           lda ,y+                         ; fetch number of bytes in this initializer
                beq LC086                       ; brif zero - we're done
                ldx ,y++                        ; get address to copy bytes to
                bsr LC04B                       ; go copy bytes
                bra LC041                       ; go handle another initializer
LC04B           ldb ,y+                         ; fetch source byte
                stb ,x+                         ; stow at destination
                deca                            ; are we done yet?
                bne LC04B                       ; brif not
                rts                             ; return to caller
LC053           pshs cc,a,b,x,y,u               ; save registers and interrupt status
                orcc #$10                       ; disable IRQ
                ldx #schedlists                 ; point to start of variables to clear for level creation
LC05A           clr ,x+                         ; clear a byte
```

```
                cmpx #hidestatus                        ; are we finished clearing things?
                blo LC05A                       ; brif not
                ldx #schedtab                   ; start of the scheduling list
                stx schedtabfree                ; mark that as the end of it
LC066           clr ,x+                         ; clear more data
                cmpx #emptyhand                 ; end of area to clear?
                blo LC066                       ; brif not
                ldy #LD7DC                      ; point to list of entries to schedule
                dec readylistchg                ; mark ready list as modified
                ldd #12                         ; set tick count to 0 and list to "ready" list
LC076           ldx ,y++                        ; get routine address
                beq LC084                       ; brif end of list
                jsr LC25C                       ; create scheduler entry
                stx 3,u                         ; save routine address in scheduling entry
                jsr LC21D                       ; add to "ready" list
                bra LC076                       ; go look for another routine
LC084           puls cc,a,b,x,y,u,pc
LC086           bsr LC053                       ; initialize new level data
                ldu #LDA91                      ; point to objects to create for game
                clra                            ; initialize object type
LC08C           ldb ,u                          ; get object info
                andb #$0f                       ; get low nibble
                stb objectcount                 ; save number to create (total)
                ldb ,u+                         ; get object info again
                lsrb                            ; get high nibble (object starting level)
                lsrb
                lsrb
                lsrb
                stb objectlevel                 ; save starting level
LC09A           createobject                    ; create an object
                dec 5,x                         ; mark as equipped or carried
                incb                            ; bump level
                cmpb #5                         ; was it level 5?
                ble LC0A5                       ; brif yes
                ldb objectlevel                 ; get level back
LC0A5           dec objectcount                 ; created enough objects?
                bne LC09A                       ; brif not
                inca                            ; move to next object type
                cmpu #LDA91+18                  ; end of table?
                blo LC08C                       ; brif not - go to next entry
                ldu #statusarea                 ; point to text parameters for status area
                dec textother                   ; indicate nonstandard text area
                clearstatus                     ; blank status line (where we'll put the copyright notice)
                renderstrimmp                   ; display copyright message
                fcb $f8,$df,$0c,$c9             ; packed string "COPYRIGHT  DYNA MICRO  MCMLXXXII"
                fcb $27,$45,$00,$02
                fcb $65,$c1,$03,$52
                fcb $39,$3c,$00,$68
                fcb $da,$cc,$63,$09
                fcb $48
                clr textother                   ; reset text rendering to standard mode
                rts                             ; transfer control to correct game loop
dodemo          dec waitnewgame                 ; flag demo game
                bsr enablepiairq                ; set up interrupts and sound
                ldx #img_wizard                 ; point to wizard image
                dec enablefadesound             ; enable fade sound effect
                fadein                          ; fade the wizard in
                renderstrimmp                   ; display <CR>"I DARE YE ENTER..."<CR>
                fcb $9f,$d2,$02,$06             ; packed "\rI DARE YE ENTER...\r" string
                fcb $45,$06,$4a,$02
                fcb $ba,$85,$97,$bd
                fcb $ef,$80
                renderstrimmp                   ; display "...THE DUNGEONS OF DAGGORATH!!!"
                fcb $f7,$bd,$ea,$20             ; packed "...THE DUNGEONS OF DAGGORATH!!!" string
                fcb $a0,$25,$5c,$72
                fcb $bd,$d3,$03,$cc
                fcb $02,$04,$e7,$7c
                fcb $83,$44,$6f,$7b
                delay                           ;* wait for about 2.6 seconds
                delay                           ;*
                fadeout                         ; fade the wizard out
                cleargfx2                       ; clear second graphics screen
                dec pageswap                    ; flag graphics swap ready
                sync                            ; wait for swap to happen
                lda #2                          ; create maze for level 3
                ldu #startobjdemo               ; point to demo game object list
                bra LC131                       ; go start demo running
enablepiairq    ldd #$343c                      ; set up initializers for PIAs
                sta PIA1+1                      ; set data mode, no interrupts, cassette off
                stb PIA1+3                      ; set data mode, no interrupts, sound on
                inca                            ; adjust to enable interrupt
                sta PIA0+3                      ; set data mode, enable VSYNC, clear MSB of analog MUX
                cwai #$ef                       ; enable IRQ and wait for one
                rts                             ; return to caller
dogame          bsr enablepiairq                ; set up interrupts and sound
                ldd #$100b                      ; maze position (11,16)
                std playerloc                   ; set start position there
                clr powerlevel                  ; reset power level to new game level (keeps LSB of default
value)
                clra                            ; create maze for level 1
                ldu #startobj                   ; point to new game backpack object list
LC131           showprepare                     ; show the PREPARE! screen
```

```
                createlevel                     ; create maze
                ldy #backpack                   ; point to backpack list head pointer
LC139           lda ,u+                         ; get object to create
                bmi LC14F                       ; brif end of list
                createobject                    ; create requested object
                inc 5,x                         ; mark object as in inventory or equipeed
                exg x,u                         ; swap object pointer and list pointer
                setobjectspecs                  ; set the specs properly (as in fully revealed)
                exg x,u                         ; swap pointers back
                clr 11,x                        ; mark object revealed
                stx ,y                          ; save new object in backpack list
                tfr x,y                         ; move list pointer to object just created
                bra LC139                       ; go look for another object to create
LC14F           tst waitnewgame                 ; are we doing a demo game?
                beq LC166                       ; brif not
                dec disablesched                ; disable scheduling events
                ldx #displayscroll              ; set to scroll display
                stx displayptr
                dec showseer                    ; set to show creatures on map
                updatedungeon                   ; show the dungeon map
                delay                           ; delay for about 2.5 seconds
                delay
                clr disablesched                ; enable scheduling events
                sync                            ; wait a couple of ticks
                sync
LC166           resetdisplay                    ; clear command and status areas and show the dungeon
                showprompt                      ; show command prompt
                jmp LC1F5                       ; go to main loop

LC16D           stx CBUFAD                      ; set address to read to
                romcall BLKIN                   ; read a block
                tsta                            ; is it the end of the file?
                lbne RESVEC                     ; brif so - premature end, fail with a reset
                ldb BLKTYP                      ; get type of block
                rts
disablepiairq   ldu #PIA0                       ; point to PIA0
                ldd #$343c                      ; set up initializers for PIA
                sta 3,u                         ; disable VSYNC interrupt, clear analogue mux msb
                sta PIA1+3                      ; disable interrupts on PIA1, cassette off
                stb PIA1+1                      ; disable interrupts on PIA1, sound on
                rts
busywait        ldx zero                        ; get long delay constant
busywait000     leax -1,x                       ; have we reached 0?
                bne busywait000                 ; brif not
                rts
LC192           bsr disablepiairq               ; set up PIA for cassette I/O
                bsr busywait                    ; delay for a bit
                bsr busywait
                romcall WRTLDR                  ; write a file header
                romcall BLKOUT
                bsr busywait                    ; delay for a bit
                romcall WRTLDR                  ; write a leader for data area
                ldx #zero                       ; point to start of game state
LC1A6           ldd #$0180                      ; set block type to data, size to 128 bytes
                std BLKTYP
                stx CBUFAD                      ; set start of buffer to write
                romcall BLKOUT                  ; write a data block
                cmpx #datatop                   ; have we reached end of state?
                blo LC1A6                       ; brif not
                stu BLKTYP                      ; write trailing block
                romcall BLKOUT
                bsr busywait                    ; delay for a bit
                bra LC1EC                       ; go init things and restart main loop
LC1C1           bsr disablepiairq               ; set up PIA for cassette I/O
                romcall CSRDON                  ; start tape
LC1C6           ldu screendraw                  ; point to drawing area
                ldx ,u                          ; get pointer to screen data - use as a read buffer
                bsr LC16D                       ; read a block
                bne LC1C6                       ; brif data block
                ldx ,u                          ; get buffer pointer
                ldu #wordbuff                   ; point to requested file name
                ldb #8                          ; 8 characters in file name
LC1D5           lda ,x+                         ; does character match?
                cmpa ,u+
                bne LC1C1                       ; brif not - look for another header
                decb                            ; end of file name?
                bne LC1D5                       ; brif not - check another
                romcall CSRDON                  ; start tape
                ldx #zero                       ; point to game state area
LC1E4           bsr LC16D                       ; read a block
                bpl LC1E4                       ; brif it was still a data block
                lds #STACK                      ; reset stack pointer
LC1EC           jsr enablepiairq                ; make sure PIAs are set right
                clr loadsaveflag                ; flag regular operations
                resetdisplay                    ; clear command and status areas, update appropriately
                showprompt                      ; show command prompt
LC1F5           ldu #schedlists+12              ; point to ready list head
                clr readylistchg                ; mark ready list restart not needed
LC1FA           tfr u,y                         ; save ready list pointer
LC1FC           tst loadsaveflag                ; are we loading or saving?
                bgt LC192                       ; brif saving
                bmi LC1C1                       ; brif loading
                ldu ,u                          ; are we at the end of the ready list?
                beq LC1F5               ; brif so
                pshs y,u                ; save registers
                jsr [3,u]               ; call the registered routine
                puls y,u                ; restore registers
                tst readylistchg        ; do we need to restart the ready list processing?
                bne LC1F5               ; brif so
                cmpb #12                ; are we leaving the routine in the ready list?
                beq LC1FA               ; brif so - check next entry
                bsr LC238               ; remove this event from the ready list
                bsr LC21D               ; reschedule in requested queue for requested number of ticks
                tfr y,u                 ; move current pointer to previous pointer
                bra LC1FC               ; go check next entry
LC21D           pshs cc,a,b,x           ; save flags and registers
                orcc #$10               ; disable IRQ
                sta 2,u                 ; reset tick count
                ldx #schedlists         ; pointer to routine base
                abx                     ; add offset
                clra                    ; NULL pointer
                clrb
                std ,u                  ; mark this timer unused
LC22B           cmpd ,x                 ; are we at a NULL pointer (end of list)?
                beq LC234               ; brif so
                ldx ,x                  ; point to next entry
                bra LC22B               ; go check if we're at the end yet
LC234           stu ,x                  ; move this timer entry to the end of the list
                puls cc,a,b,x,pc        ; restore registers, interrupt status,  and return
LC238           pshs cc,x               ; save interrupt status and registers
                orcc #$10               ; disable IRQ
                ldx ,u                  ; get next pointer from this entry
                stx ,y                  ; save it in previous entry
                puls cc,x,pc            ; restore interrupts, registers, and return
LC242           pshs b,x,y,u            ; save registers
                tst disablesched        ; are we handling timers?
                bne LC25A               ; brif not
LC248           tfr u,y                 ; save timer pointer
                ldu ,u                  ; get pointer to timer info
                beq LC25A               ; brif nothing doing for timer
                dec 2,u                 ; has this timer record expired?
                bne LC248               ; brif not - check next one
                bsr LC238               ; go process timer record
                ldb #12                 ; offset to "ready" list
                bsr LC21D               ; go move event entry to "ready" list
                bra LC248               ; go process next timer record
LC25A           puls b,x,y,u,pc         ; restore registers and return
LC25C           pshs x                  ; save registers
                ldu schedtabfree        ; get open slot for scheduling
                leax 7,u                        ; point to next open slot
                stx schedtabfree                ; save new open slot for scheduling
                puls x,pc                       ; restore registers and return
; Set the SAM video mode and display offset register to the value in D. Starting at the lsb of
; D, the SAM bits are programmed from FFC0 upward. This sets bits 9-0 of the SAM register
; to match the value in D.
setSAM          pshs x,b,a                      ; save registers
                ldx #SAMREG                     ; point to SAM register
setSAM000       lsra                            ;* shift the bit value to set to carry
                rorb                            ;*
                bcs setSAM001                   ; brif bit set
                sta ,x                          ; clear the bit
                skip2                           ; skip next instruction
setSAM001       sta 1,x                         ; set the bit
                leax 2,x                        ; move to next SAM register bit
                cmpx #SAMREG+$14                ; are we at the end of the register?
                blo setSAM000                   ; brif not
                puls a,b,x,pc                   ; restore registers and return
; IRQ service routine
irqsvc          ldx #PIA1                       ; point to PIA1
                lda -29,x                       ; get interrupt status
                lbpl LC320                      ; brif not VSYCN
                lda #zero/256                   ; point to direct page MSB
                tfr a,dp                        ; make sure DP is set correctly
                tst pageswap                    ; do we have a screen swap to do?
                beq LC29D                       ; brif not
                ldd screenvis                   ; get currently visible screen pointer
                ldu screendraw                  ; get newly drawn screen pointer
                std screendraw                  ; save current screen as screen to draw
                stu screenvis                   ; save drawn screen as current
                ldd 4,u                         ; get the SAM value for the new screen
                bsr setSAM                      ; go program the SAM
                clr pageswap                    ; flag no swap needed
LC29D           tst dofadesound                 ; are we doing the "fade buzz" thing?
                beq LC2A9                       ; brif not
                com fadesoundval                ; invert the bits of of the value
                lda fadesoundval                ; fetch new value
                lsla                            ;* align to DAC bits
                lsla                            ;*
                sta ,x                          ; set DAC output
LC2A9           tst enableheart                 ; is the heart beating?
                beq LC2DC                       ; brif not
                dec heartctr                    ; count down ticks till beat
                bne LC2DC                       ; brif not expired
                lda hearticks                   ; fetch ticks till next beat
                sta heartctr                    ; reset counter
                ldb 2,x                         ; fetch single bit sound register
                eorb #2                         ; invert single bit sound output
                stb 2,x                         ; set new sound output
                tst hidestatus                  ; is the status line shown?
                beq LC2DC                       ; brif not - don't update heart
                ldu #statusarea                 ; point to status line text area descriptor
                ldx 4,u                         ; fetch current text position
                ldd #15                         ; position for centring heart
                std 4,u                         ; save output position
                lda #$20                        ; code for contracted heart (left)
                com heartstate                  ; invert heart state
                beq LC2D1                       ; brif contracted
                lda #$22                        ; code for expanded heart (left)
LC2D1           jsr LCA17                       ; render left half of heart
                inc 5,u                         ; bump character position
                inca                            ; code for right half of heart
                jsr LCA17                       ; render right half of heart
                stx 4,u                         ; save original text position
LC2DC           ldu #schedlists+2               ; point to timer lists
                jsr LC242                       ; check if any records expired at 60Hz
                ldx #clockctrs                  ; point to timer records
                ldy #LC324                      ; point to timer max values
LC2E9           inc ,x                          ; bump timer value
                cmpx #clockctrs+5               ; end of timer record?
                beq LC2FF                       ; brif so
                lda ,x                          ; fetch new value
                cmpa ,y+                        ; has timer maxed out?
                blt LC2FF                       ; brif not
                clr ,x+                         ; reset timer record
                leau 2,u                        ; move to next timer list
                jsr LC242                       ; see if any events have expired
                bra LC2E9                       ; go handle next timer
LC2FF           tst nokeyboard                  ; are we polling the keyboard?
                bne LC320                       ; brif not
                tst waitnewgame                 ; are we running a demo/waiting for keypress for game start?
                beq LC318                       ; brif not
                clr PIA0+2                      ; strobe all keyboard columns
                lda PIA0                        ; fetch row data
                anda #$7f                       ; mask off comparator input
                cmpa #$7f                       ; did we have any keys down?
                beq LC320                       ; brif not
                ldx #LC005                      ; pointer to game start routine
                stx 10,s                        ; set return to game start routine
LC318           romcall POLCAT                  ; poll the keyboard
                tsta                            ; was a key down?
                beq LC320                       ; brif not
                bsr writekeybuf                 ; go process keyboard input
LC320           lda PIA0+2                      ; clear interrupt status
                rti
; These are the rollover points for the timers. Each timer only ticks if the previous
; timer has rolled over.
LC324           fcb 6                           ; tick 10 times per second
                fcb 10                          ; tick 1 time per second
                fcb 60                          ; tick 1 time per minute
                fcb 60                          ; tick 1 time per hour
                fcb 24                          ; tick 1 time per day
readkeybuf      pshs cc,b,x                     ; save registers and interrupt status
                orcc #$10                       ; disable IRQ
                clra                            ; flag no key pressed
                ldx #keybuf                     ; point to keyboard ring buffer
                ldb keybufread                  ; get buffer read offset
                cmpb keybufwrite                ; same as buffer write offset?
                beq readkeybuf000               ; brif so - no key available
                lda b,x                         ; fetch key from buffer
                incb                            ; bump buffer pointer
                andb #$1f                       ; wrap around if needed
                stb keybufread                  ; save new buffer read offset
readkeybuf000   puls cc,b,x,pc                  ; restore registers and interrupts
; Add a keypress to the keyboard buffer. NOTE: this does not check for buffer overflow
; which means when the buffer gets full, it just rolls over and overwrites the previous
; data.
writekeybuf     pshs cc,b,x                     ; save registers and interrupt status
                orcc #$10                       ; disable IRQ
                ldx #keybuf                     ; point to keyboard ring buffer
                ldb keybufwrite                 ; get buffer write offset
                sta b,x                         ; stash new key
                incb                            ; bump buffer pointer
                andb #$1f                       ; wrap around if needed
                stb keybufwrite                 ; save new buffer write offset
                puls cc,b,x,pc                  ; restore registers and interrupts
; SWI handler
swisvc          andcc #$ef                      ; re-enable IRQ - SWI disables it
                ldx 10,s                        ; get return address
                lda ,x+                         ; get operation code
                stx 10,s                        ; save new return address
                ldx #LC384                      ; point to first SWI routine
                ldu #LC995                      ; point to routine offset table
LC360           ldb ,u+                         ; get length of previous routine
                abx                             ; add to routine pointer
                deca                            ; are we at the right routine?
                bpl LC360                       ; brif not
                stx ,--s                        ; save routine address
                ldd 3,s                         ; restore D register
                ldx 6,s                         ; restore X register
                ldu 10,s                        ; restore U register
                jsr [,s++]                      ; call the routine
                rti                             ; return to caller
; SWI2 handler
swi2svc         clrb                            ;* restore direct page for ROM call
                tfr b,dp                        ;*
                ldu 10,s                        ; get return address
                ldb ,u+                         ; get ROM routine offset
                stu 10,s                        ; save new return address
                ldu #ROMTAB                     ; point to ROM vector table
                jsr [b,u]                       ; call the ROM routine
                sta 1,s                         ;* save return values
                stx 4,s                         ;*
                rti                             ; return to caller
; SWI 0 routine
; Calculate base light level in dungeon. Note that "magic lighting" is also used for simulating
; the fadeout and fade in during fainting.
LC384           lda effectivelight              ; fetch effective light level in dungeon
                tst rendermagic                 ; are we checking for special lighting conditions?
                beq LC38E                       ; brif not
                lda effectivemlight             ; get magical light level
                clr rendermagic                 ; undo special light level checking
LC38E           clrb                            ; default to full bright
                suba #7                         ; adjust level based on the order of the table used
                suba renderdist                 ; subtract render distance from light level
                bge LC39F                       ; brif adjusted light level >= 0 - means we can see everything
                decb                            ; change to dark default
                cmpa #$f9                       ; are we in a partial visible range?
                ble LC39F                       ; brif not - use the default value (dark)
                ldx #LCB96                      ; point to end of table of pixel masks, used for powers of two
levels
                ldb a,x                         ; fetch value from pixel mask (1, 2, 4, 8, 16, 32)
LC39F           stb lightlevel                  ; save new light level (full bright, dark, or partial)
                rts                             ; return to caller
; SWI 1 routine
;***************************************************************************************************
; This routine renders a line graphic from the specification stored at (X).
;
; The data at (X) is a series of operations as follows:
;
; if (X) is < $FA, then the two bytes at (X) are an absolute Y and X coordinate. If polyfirst is clear, this
is
; the first vertex in a polygon and the coordinates are simply recorded. Otherwise, a line is drawn from
; the previous coordinates to the new coordinates. These coordinates have the Y coordinate first.
```

```
;
; If (X) is >= $FA, it is a special operation defined as follows:
;
; FA: return from a "subroutine" to the previous flow
; FB: call a subroutine at the memory address in the next two bytes
; FC: draw a series of points using relative motion. The following byte is split into nibbles with the
;       upper nibble being the Y displacement and the lower nibble being the X displacement. These values
;       are signed and will be doubled when applied to the drawing. This gives a range of -32 to +30 in
;       steps of 2 for each direction. If both displacements are zero (a zero byte), this is the end of
;       the relative sequence. The end of one of these sequences is the end of a polygon.
; FD: like FB but doesn't record the previous location
; FE: flags the end of the input and causes a return to the caller. Do not use this in a subroutine as
;       the stack will have been used to record the return data location.
; FF: mark the next coordinates as the start of a new polygon.
;
; In all cases, the X and Y coordinates actually used have a scale factor applied to them based on the
; distance from the defined centre of the graphics area which is stored in (horizcent,vertcent). The
horizontal
; scale factor is at horizscale and the vertical scaling factor is at vertscale. A factor of 128 serves as a
scale
; factor of 1. 192 would be 1.5 and 64 would be 0.5.
;
; Variables used:
;
; horizcent     the horizontal centre point for rendering graphics and scaling
; vertcent      the vertical centre point for rendering graphics and scaling
; lightlevel    the light level with respect to rendering the graphic
; horizscale    the horizontal scaling factor (binary point to the right of bit 7)

; vertscale     the vertical scaling factor (binary point to the right of bit 7)
; polyfirst     nonzero if this is not the first coordinate in a polygon
; lastunscalex  the most recent absolute unscaled X coordinate
; lastunscaley  the most recent absolute unscaled Y coordinate
LC3A2           clr polyfirst                   ; mark input as start of polygon
                lda lightlevel                  ; fetch dungeon light level
                inca                            ; is it $ff (dark)?
                beq LC3F6                       ; brif so - skip rendering
LC3A9           ldb ,x                          ; fetch input data
                subb #$fa                       ; adjust for operation codes
                blo LC3CF                       ; brif not operation code
                leax 1,x                        ; move on to next image data byte
                ldy #LC3B9                      ; point to jump table for operation codes
                ldb b,y                         ; get offset to operation routine
                jmp b,y                         ; execute operation routine
LC3B9           fcb LC3C9-LC3B9                 ; (FA) return from a "subroutine"
                fcb LC3BF-LC3B9                 ; (FB) call a "subroutine"
                fcb LC417-LC3B9                 ; (FC) polygon
                fcb LC3C6-LC3B9                 ; (FD) jump to a new "routine"
                fcb LC3F6-LC3B9                 ; (FE) end of input - return to caller
                fcb LC3CB-LC3B9                 ; (FF) next coordinates are start of new polygon
LC3BF           ldd ,x++                        ; get address of "subroutine" to call
                stx ,--s                        ; save return address
                tfr d,x                         ; set new "execution" address
                skip2                           ; skip next instruction
LC3C6           ldx ,x                          ; get address of "routine" to jump to
                skip2                           ; skip next instruction
LC3C9           ldx ,s++                        ; get back saved input location
LC3CB           clr polyfirst                   ; reset polygon start flag to start
                bra LC3A9                       ; go process more input
LC3CF           tst polyfirst                   ; is this the first coordinate in a polygon?
                bne LC3D9                       ; brif not
                bsr LC3E2                       ; fetch input coordinates and save them
                dec polyfirst                   ; flag as not first coordinate
                bra LC3A9                       ; go process more input
LC3D9           bsr LC3E0                       ; set up coordinates to draw a line
                jsr drawline                    ; draw the line
                bra LC3A9                       ; go process more input
LC3E0           bsr LC3F7                       ; move last end coordinates to line start
LC3E2           ldb ,x+                         ; get the next Y coordinate and move pointer forward
                stb lastunscaley                ; save unscaled Y coordinate
                bsr LC400                       ; scale the Y coordinate
                addd vertcent                   ; add in base Y coordinate
                std yend                        ; save scaled end coordinate for line
                ldb ,x+                         ; get the next X coordinate and move pointer forward
                stb lastunscalex                ; save unscaled X coordinate

                bsr LC406                       ; scale the X coordinate
                addd horizcent                  ; add in base X coordinate
                std xend                        ; save scaled X coordinate for line
LC3F6           rts                             ; return to caller
LC3F7           ldd yend                        ; fetch last Y coordinate
                std ybeg                        ; save as begining of new line segment
                ldd xend                        ; fetch last X coordinate
                std xbeg                        ; save as beginning of new line segment
                rts                             ; return to caller
LC400           lda vertscale                   ; get desired vertical scaling factor
                subb vertcent+1                 ; find difference from Y base coordinate
                bra LC40A                       ; go finish calculating scale
LC406           lda horizscale                  ; get desired horizontal scale factor
                subb horizcent+1                ; find difference from X base coordinate
LC40A           bcs LC40F                       ; brif negative difference
                mul                             ; apply the scaling factor
                bra LC414                       ; normalize to an integer in D and return
LC40F           negb                            ; make coordinate difference positive
```

```
                mul                             ; apply the scaling factor
                jsr LCA99                       ; negate coordinate value
LC414           jmp asrd7                       ; normalize to an integer in D and return
LC417           lda ,x+                         ; get next byte in input
                beq LC3CB                       ; brif NUL - end of values
                bsr LC3F7                       ; move last end coordinate to start coordinate for line
                ldb -1,x                        ; get the relative movement specifications
                asrb                            ;* fetch high nibble signed extended into B
                asrb                            ;*
                asrb                            ;*
                asrb                            ;*
                lslb                            ; and multiply by two
                addb lastunscaley               ; add in previous Y coordinate
                stb lastunscaley                ; save new Y coordinate
                bsr LC400                       ; go scale the Y coordinate
                addd vertcent                   ; add in the Y base coordinate
                std yend                        ; save new ending Y coordinate
                ldb -1,x                        ; get back the input byte again
                andb #$0f                       ; mask off the upper bits
                bitb #8                         ; is bit 3 set?
                beq LC438                       ; brif not
                orb #$f0                        ; sign extend to 8 bits
LC438           lslb                            ; multiply by two
                addb lastunscalex               ; add in saved X coordinate
                stb lastunscalex                ; save new X coordinate
                bsr LC406                       ; go scale the X coordinate
                addd horizcent                  ; add in base X coordinate
                std xend                        ; save new ending X coordinate
                jsr drawline                    ; go draw a line
                bra LC417                       ; look for another line segment
; swi 2 routine
; fetch a packed string immediately following the call and display it
LC448           ldx 12,s                        ; fetch return address - string address
                decodestrsb                     ; go decode string
                stx 12,s                        ; save new return address - after string
                ldx #stringbuf                  ; point to decoded string
                skip2                           ; skip the next instruction - nothing to display yet
LC452           renderchar                      ; display character in A
; swi 3 routine
; display an unpacked string pointed to by X
LC454           lda ,x+                         ; fetch byte from string
                bpl LC452                       ; brif not end of string - display it
                rts                             ; return to caller
; swi 4 routine
; display character in A
LC459           tst textother                   ; are we looking for standard text mode?
```

```
                bne LC460                       ; brif not
                ldu #commandarea                ; point to display state information
LC460           ldx 4,u                         ; fetch current screen location
                jsr LC9B2                       ; actually display the appropriate character
                cmpx 2,u                        ; are we at the end of text area?
                blo LC46C                       ; brif not
                jsr LC9D4                       ; go scroll the text area
LC46C           stx 4,u                         ; save new screen location
                rts                             ; return to caller
; swi 5 routine - decode packed string at X to stringbuf
LC46F           ldu #stringbuf                  ; point to output buffer
; swi 6 routine - decode a packed string at X to U
; the first value is the length of the string less one
LC472           leay -1,u                       ; point to working data before buffer
                clr ,y                          ; initialize value counter
                bsr LC48C                       ; fetch a value
                tfr b,a                         ; save length
LC47A           bsr LC48C                       ; fetch a value
                stb ,u+                         ; save in output
                deca                            ; done yet?
                bpl LC47A                       ; brif not
                sta ,u                          ; flag end of string with $ff
                tst ,y                          ; did we consume an even number of bytes?
                beq LC489                       ; brif so
                leax 1,x                        ; move pointer forward
LC489           stx 6,s                         ; save pointer past end of input
                rts                             ; return to caller
LC48C           pshs a,u                        ; save registers
                lda ,y                          ; get value counter
                ldu #LC4A2                      ; point to value handlers
                lda a,u                         ; get offset to handler for this value
                jsr a,u                         ; call the handler for this value
                lda ,y                          ; get value counter
                inca                            ; bump it
                anda #7                         ; wrap it around - the pattern repeats every 8 values
                sta ,y                          ; save new value counter
                andb #$1f                       ; values are only 5 bits - clear out extra bits
                puls a,u,pc                     ; restore registers and return
LC4A2           fcb LC4AA-LC4A2                 ; value 0 handler
                fcb LC4B0-LC4A2                 ; value 1 handler
                fcb LC4B5-LC4A2                 ; value 2 handler
                fcb LC4B9-LC4A2                 ; value 3 handler
                fcb LC4BE-LC4A2                 ; value 4 handler
                fcb LC4C3-LC4A2                 ; value 5 handler
                fcb LC4C7-LC4A2                 ; value 6 handler
                fcb LC4CC-LC4A2                 ; value 7 handler
```

```
; value 0: upper 5 bits of current input byte
LC4AA           ldb ,x                          ; fetch input byte
                lsrb                            ;* align in low bits of B
LC4AD           lsrb                            ;*
LC4AE           lsrb                            ;*
                rts                             return to caller
; value 1: lower 3 bits of current input byte and upper 2 bits of next one
; consumes a byte
LC4B0           ldd ,x+                         ; fetch input data and consume a byte
                jmp asrd6                       ; align in low bits of B
; value 2: bits 5...1 of current input byte
LC4B5           ldb ,x                          ; fetch input byte
                bra LC4AE                       ; align in low bits of B
; value 3: bits 0 of current byte and upper 4 bits of next one
; consumes a byte
LC4B9           ldd ,x+                         ; fetch input data and consume a byte
                jmp asrd4                       ; align in low bits of B
; value 4: low 4 bits of input byte and high bit of next one
; consumes a byte
LC4BE           ldd ,x+                         ; fetch input data and consume a byte
                jmp asrd7                       ; align in low bits of B
; value 5: bits 6...2 of current input byte
LC4C3           ldb ,x                          ; fetch input data
                bra LC4AD                       ; align in low bits of B
; value 6: low two bits of current input byte and high 3 bits of next one
; consums a byte
LC4C7           ldd ,x+                         ; fetch input data and consume a byte
                jmp asrd5                       ; align in low bits of B
; value 7: low 5 bits of current input byte
; consumes a byte
LC4CC           ldb ,x+                         ; fetch input data - already aligned
                rts                             ; return to caller
; swi 7 routine
; Generate a pseudo random number based on seed in randomseed, return 8 bit value in A
LC4CF           ldx #8                          ; need to generate 8 bits
LC4D2           clrb                            ; initialize 1s counter
                ldy #8                          ; 8 bits in byte to count
                lda randomseed+2                ; get lsb of seed
                anda #$e1                       ; drop bits 4-1 (keep 7,6,5,0)
LC4DB           lsla                            ; shift modified seed lsb left
                bcc LC4DF                       ; brif no carry
                incb                            ; bump 1s count
LC4DF           leay -1,y                       ; done 8 bits?
                bne LC4DB                       ; brif not
                lsrb                            ; take bit 0 of the count
                rol randomseed                  ;* and shift it into the seed value
                rol randomseed+1                ;*
                rol randomseed+2                ;*
                leax -1,x                       ; have we generated 8 bits?
                bne LC4D2                       ; brif not
                lda randomseed                  ; get msb of current seed value
                sta 3,s                         ; save 8 bit random value for return
                rts                             ; return to caller
; swi 8 routine - clear first graphics screen
LC4F3           ldu screenvis                   ; point to first screen parameter block
                skip2                           ; skip next instruction
; swi 9 routine - clear second graphics screen
LC4F6           ldu screendraw                  ; point to second screen parameter block
                ldb levbgmask                   ; get current level background colour
                bsr LC517                       ; go clear the graphics area of the screen
                stu 10,s                        ; save pointer to parameter block for the caller
                rts                             ; return to caller
; swi 10 routine - clear the status line
LC4FF           ldx #statusarea                 ; point to text area parameters for the status line
                ldu #LD87C                      ; point to screen address table for the status line
                bra LC50D                       ; go clear the status line
; swi 11 routine - clear the command entry area
LC507           ldx #commandarea                ; point to text area parameters for the command area
                ldu #LD888                      ; point to screen address table for the command area
LC50D           clr 4,x                         ;* set current cursor to start of text area
                clr 5,x                         ;*
                ldb 6,x                         ; get background colour of text area
                bsr LC517                       ; go clear text area
                leau 6,u                        ; and repeat the process for the other graphics screen
LC517           pshs a,b,x,y,u                  ; save regsiters
                sex                             ; get background colour to A
                tfr d,y                         ; move it into Y too (4 bytes of background colour)
                leax ,u                         ; point to start of parameter area
                ldu 2,u                         ; get address of end of text area (+1)
LC520           pshu a,b,y                      ; blast 4 background bytes to area
                cmpu ,x                         ; are we at the start of the area?
                bne LC520                       ; brif not
                puls a,b,x,y,u,pc               ; restore registers and return
; swi 12 routine
; Check for fainting or recovery from damage and handle the fading out and fading in as a result.
; Also check for death due to damage level exceeding power level.
LC529           clr accum0                      ; mark high bits of 24 bit accumulator
                ldd powerlevel                  ; get current power level
                std accum0+1                    ; save it in accumulator
                lda #6                          ; shift left 6 bits (times 64)
LC531           lsl accum0+2                    ;* do one left shift
                rol accum0+1                    ;*
                rol accum0                      ;*
                deca                            ; done enough shifts?
                bne LC531                       ; brif not
                clr accum1                      ; clear high bits of 24 bit accumulator
                ldd damagelevel                 ; get damage level
                std accum1+1                    ; stow in accumulator
                lsl accum1+2                    ;* shift left (times 2)
                rol accum1+1                    ;*
                rol accum1                      ;*
                ldd powerlevel                  ; get current power level
                addd accum1+1                   ; add in half damage level
                std accum1+1                    ; save low word
                ldb accum1                      ;* propagate carry
                adcb #0                         ;*
                stb accum1                      ;*
                clr accum2                      ; initialize quotient
LC554           ldd accum0+1                    ; get low bits of powerlevel/64
                subd accum1+1                   ; subtract (powerlevel + damagelevel * 2)
                std accum0+1                    ; save low word
                lda accum0                      ; fetch msb of powerlevel/64
                sbca accum1                     ; finish subtracting with msb of (powerlevel + damagelevel *
2)
                sta accum0                      ; save it in msb of result
                inc accum2                      ; bump quotient
                bcc LC554                       ; brif no carry from addition - we haven't got a result yet
                lda accum2                      ; get division result
                suba #19                        ; subtract 19


                sta hearticks                   ; save number of ticks before redoing the calculation and also
how fast heart beats
                tst nokeyboard                  ; are we blocking the keyboard?
                bne LC595                       ; brif so
                cmpa #3                         ; is number of ticks > 3?
                bgt LC5AE                       ; brif so
                clearcommand                    ; clear the command area
                lda effectivelight              ; fetch the effective light level
                sta savedefflight               ; save it
LC578           dec effectivemlight             ; mark us as passed out
                jsr [displayptr]                ; update the main display area
                dec pageswap                    ; set graphics swap required
                sync                            ; wait for swap to happen
                dec effectivelight              ; reduce effective light level
                lda effectivelight              ; fetch new light level
                cmpa #$f8                       ; have we reached a minimum?
                bgt LC578                       ; brif not
                cleargfx2                       ; clear graphics
                dec pageswap                    ; set graphics swap required
                dec nokeyboard                  ; disable keyboard
                clr keybufread                  ;* reset keyboard buffer - we passed out so clear any commands
                clr keybufwrite                 ;*
                bra LC5AE                       ; get on with things
LC595           cmpa #4                         ; have we recovered enough to wake up?
                ble LC5AE                       ; brif not
LC599           jsr [displayptr]                ; update the main display area
                dec pageswap                    ; set graphics swap required
                sync                            ; wait for swap to happen
                inc effectivemlight             ; mark as not passed out
                inc effectivelight              ; bump effective light level
                lda effectivelight              ; fetch new light level
                cmpa savedefflight              ; are we at old intensity?
                ble LC599                       ; brif not
                clr nokeyboard                  ; re-enable keyboard
                showprompt                      ; show the prompt
LC5AE           ldx powerlevel                  ; get current power level
                cmpx damagelevel                ; is it less than damage level?
                blo LC5B5                       ; brif so - we're dead!
                rts                             ; returnt o caller
; This routine handles player death
LC5B5           ldx #img_wizard                 ; point to wizard image
                dec enablefadesound             ; neable fade sound effect
                fadeinclrst                     ; fade in the wizard
                renderstrimmp                   ; display "YET ANOTHER DOES NOT RETURN..."
                fcb $ff,$c1,$92,$d0             ; packed "YET ANOTHER DOES NOT RETURN..." string
                fcb $01,$73,$e8,$82
                fcb $c8,$04,$79,$66
                fcb $07,$3e,$80,$91
                fcb $69,$59,$3b,$de
                fcb $f0
                clr nokeyboard                  ; enable keyboard polling in IRQ
                dec waitnewgame                 ; set up to wait for keypress to start new game
LC5D7           bra LC5D7                       ; wait forever (or until the IRQ does something)
; swi 13 routine
LC5D9           ldu #statusarea                 ; point to parameters for status line
                dec textother                   ; set to nonstandard text area
                lda levbgmask                   ; get current level background
                coma                            ; invert it for status line
                sta 6,u                         ; set up for displaying status line
                clra                            ; set position to start clearing (start of line)
                clrb
                bsr LC609                       ; clear half the line
                std 4,u                         ; save display position
                ldx lefthand                    ; fetch object in left hand
                bsr LC617                       ; get name of object
                renderstr                       ; display left hand object
                ldd #17                         ; set position to start clearing
                bsr LC609                       ; go clear half the line
                ldx righthand                   ; fetch object in right hand
                bsr LC617                       ; get name of object
                tfr x,y                         ; save start pointer
                ldd #$21                        ; set up offset for displaying right justified
LC5FD           decb                            ; move cursor point left
                tst ,y+                         ; end of string yet?
                bpl LC5FD                       ; brif not - keep moving left
                std 4,u                         ; save render position
                renderstr                       ; display the right hand object
                clr textother                   ; reset to standard text rendering
                rts
LC609           pshs a,b                        ; save registers
                std 4,u                         ; save the start position
                ldd #15                         ; set up for a space (code 0) 15 times
LC610           renderchar                      ; render a space
                decb                            ; done yet?
                bne LC610                       ; brif not
                puls a,b,pc                     ; restore registers and return
LC617           pshs a,b,y,u                    ; save registers
                leay ,x                         ; point to object data
                bne LC622                       ; brif there is object data
                ldx #LC650                      ; point to "EMPTY" string
                bra LC63C                       ; return result
LC622           ldu #wordbuff                   ; point to word buffer
                tst 11,y                        ; has it been revealed?
                bne LC632                       ; brif not
                lda 9,y                         ; fetch sub type
                ldx #kw_supreme                 ; point to first "adjective" keyword
                bsr LC63E                       ; copy correct string into buffer
                clr -1,u                        ; make a space after adjective
LC632           lda 10,y                        ; get base type
                ldx #kw_flask                   ; point to first base type keyword
                bsr LC63E                       ; copy correct string into buffer
                ldx #wordbuff                   ; point to start of string
LC63C           puls a,b,y,u,pc                 ; restore registers and return
LC63E           pshs a,x                        ; save registers
LC640           decodestrsb                     ; decode the current string into buffer
                deca                            ; are we there yet?
                bpl LC640                       ; brif not
                ldx #stringbuf+1                ; point to actual string (past object type)
LC648           lda ,x+                         ; fetch character from decoded keyword
                sta ,u+                         ; save in output buffer
                bpl LC648                       ; brif not end of string yet
                puls a,x,pc                     ; restore registers and return
LC650           fcb $05,$0d,$10,$14,$19,$ff     ; unpacked string "EMPTY"
; swi 14 routine
LC656           tst nokeyboard                  ; is keyboard disabled?
                bne LC65F                       ; brif so - return, don't update display
                bsr LC660                       ; go update the display
                dec pageswap                    ; flag graphics swap required
                sync                            ; wait for swap to happen
LC65F           rts
LC660           pshs a,b,x,y,u                  ; save registers
                ldd baselight                   ; get dungeon base lighting
                ldu curtorch                    ; is there a torch lit?
                beq LC66C                       ; brif not
                adda 7,u                        ; add in physical light from torch
                addb 8,u                        ; add in magical light from torch
LC66C           std effectivelight              ; save effective light level for dungeon
                jsr [displayptr]                ; update the main display area
                puls a,b,x,y,u,pc
; swi 15 routine
LC674           ldx #LC67A                      ; point to newline followed by prompt
                renderstr                       ; go display the newline and prompt
                rts                             ; return to caller
LC67A           fcb $1f,$1e                     ; unpacked string CR PERIOD UNDERSCORE BS (including 
following)
LC67C           fcb $1c,$24,$ff                 ; unpacked string UNDERSCORE BS

; swi 16 routine

; delay for 81 ticks (1.3 seconds)
LC67F           ldb #$51                        ; fetch delay tick count
LC681           sync                            ; wait for a tick
                decb                            ; are we done yet?
                bne LC681                       ; brif not
                rts
; these two routine clear an area to 0 (black) or $ff (white) starting at X and
; ending at U
; swi 17 routine
LC686           clra                            ; set area to $00 (clear to black)
                skip2                           ; skip next byte
; swi 18 routine
LC688           lda #$ff                        ; set area to $FF (clear to white)
LC68A           sta ,x+                         ; clear a byte
                cmpx 10,s                       ; are we done yet?
                bne LC68A                       ; brif not
                rts                             ; return to caller
; This looks like a leftover from earlier development which had the
; rom calls as a SWI call instead of using SWI2. This routine cannot
; be reached through the SWI mechanism and it cannot be called directly
LC691           clrb                            ;* reset direct page for ROM call
                tfr b,dp                        ;*
                ldu 12,s                        ; fetch return address
                ldb ,u+                         ; fetch rom call wanted
                stu 12,s                        ; save new return address
                ldu #ROMTAB                     ; point to ROM vector table
                jsr [b,u]                       ; call the routine
                sta 3,s                         ;* save return values
                stx 6,s                         ;*
                rts
; swi 19 routine
; fade in the image specified by (X) with sound effects, clear status line and command area
LC6A4           clr enableheart                 ; disable heartbeat
                clearstatus                     ; clear the status area
; swi 20 routine
; fade in the image specified by (X) with sound effects, clear command area
LC6A8           clearcommand                    ; clear the command area
                ldd #$8080                      ;* set X and Y scale values to 1.0
                std horizscale                  ;*
                ldb enablefadesound             ; are we doing sound effects on the fade?
                beq LC6B7                       ; brif not
                ldb #$20                        ; set apparent lighting to 32 (less apparent)
                dec dofadesound                 ; enable fade sound effect
LC6B7           bsr LC6D7                       ; go draw the image
                decb                            ;* reduce lighting count - make more apparent
                decb                            ;*
                bpl LC6B7                       ; brif not done 16 steps
                clr dofadesound                 ; disable fade sound effect
                clr enablefadesound             ; turn off fade sound effect
LC6C1           playsoundimm $16                ; play sound effect
                rts                             ; return to caller
; swi 21 routine
; fade out the image specified by (X) with sound effects, clear command area
LC6C5           clearcommand                    ; clear the command entry area
                bsr LC6C1
                clrb                            ; set apparent illumination to fully lit
                dec dofadesound                 ; enable the fade buzz sound effect
LC6CC           bsr LC6D7                       ; go draw the image
                incb                            ;* bump lighting count (make less apparent)
                incb                            ;*
                cmpb #$20                       ; have we done 16 steps?
                bne LC6CC                       ; brif not
                clr dofadesound                 ; disable the fade buzz sound effect
                rts                             ; return to caller
```

```
LC6D7           pshs x,u                        ; save registers
                stb lightlevel                  ; set illumination value for graphic rendering
                stb fadesoundval                ; save intensity for the fade sound
                cleargfx2                       ; clear second graphics screen
                drawgraphic                     ; go draw graphic
                dec pageswap                    ; flag graphics swap required
                sync                            ; wait for swap to happen
                puls x,u,pc                     ; restore registers and return
; swi 22 routine - display the PREPARE! screen
LC6E6           jsr LD489                       ; clear second graphics screen and set up for text mode
                ldd #$12c                       ;* set cursor position to the middle of the screen
                std 4,u                         ;*
                renderstrimmp                   ; display the PREPARE! message
                fcb $3c,$24,$58,$06             ; packed string "PREPARE!"
                fcb $45,$d8
                clr textother                   ; reset to standard text rendering
                dec pageswap                    ; set graphic swap required
                rts                             ; return to caller
; swi 23 routine
; Create a new object. Associate it with the level number in B. Object type in A.
LC6FB           ldu objectfree                  ; fetch free point in object table
                stu 6,s                         ; save pointer for return
                leax 14,u                       ; move to next entry in table
                stx objectfree                  ; save as new free point in object table
                sta 9,u                         ; set object type to requested type
                stb 4,u                         ; set object level
                setobjectspecs                  ; set up object specs from data tables
                ldb 10,u                        ; fetch object general type
                ldx #LC719                      ; point to modifier table
                lda b,x                         ; get modified type entry
                bmi LC718                       ; brif no modification
                ldb 11,u                        ; get reveal strength of original object type
                setobjectspecs                  ; set up object data from replacement type
                stb 11,u                        ; restore reveal strength
LC718           rts
LC719           fcb $ff                         ; flasks do not get a replacment
                fcb $ff                         ; rings do not get a replacement
                fcb $ff                         ; scrolls do not get a replacement
                fcb $10                         ; shields default to leather shield specs
                fcb $11                         ; swords default to wooden sword specs
                fcb $0f                         ; torches default to pine torch specs
; swi 24 routine
LC71F           lsla                            ; four bytes per object specs entry
                lsla
                ldx #objspecs                   ; point to object data table
                leay a,x                        ; point to correct entry in table
                leax 10,u                       ; point to location in data table
                lda #4                          ; four bytes to copy
                jsr LC04B                       ; copy data into new object
                ldx #objextraspecs-4            ; point to extra object data
LC730           leax 4,x                        ; move to next entry
                lda ,x                          ; is it end of table?
                bmi LC742                       ; brif so
                cmpa 3,s                        ; is this entry for the object type we're creating?
                bne LC730                       ; brif not - try another
                ldd 1,x                         ; copy the ring charges and defensive values
                std 6,u
                lda 3,x
                sta 8,u
LC742           rts                             ; return to caller
; swi 25 routine
LC743           clearstatus                     ; clear the status line
                clearcommand                    ; clear the command area
                checkdamage                     ; update damage information
                inc heartctr                    ; bump count until next heart beat
                dec hidestatus                  ; set command processing to proceed normally
                dec enableheart                 ; enable heartbeat
                updatestatus                    ; update status line to current information
cmd_look        ldx #LCE66                      ; standard dungeon view routine
                stx displayptr                  ; restore display to standard dungeon view
                updatedungeon                   ; update display
                rts                             ; return to caller
; swi 26 routine
LC759           sta currentlevel                ; save current dungeon level
                ldb #12                         ; number of entries in creature count table
                mul                             ; calculate offset to creature counts for this level
                addd #creaturecounts            ; point to correct creature count table for this level
                std creaturecntptr              ; save pointer to creature count table
                ldb currentlevel                ; get back current level number
                ldx #holetab                    ; point to hole/ladder table
LC768           stx holetabptr                  ; save hole/ladder data pointer
LC76A           lda ,x+                         ; fetch flag
                bpl LC76A                       ; brif we didn't consume a flag
                decb                            ; are we at the right set of data for the level?
                bpl LC768                       ; brif not - save new pointer and search again
                ldx #creaturetab                ; get start address to clear
                ldu #mazedata                   ; get end address to clear
                clearblock                      ; go clear area to zeros
                jsr LC053                       ; initialize data for new level
                jsr createmaze                  ; create the maze
                ldu creaturecntptr              ; point to creature counts for this level
                lda #11                         ; offset for wizard
```

```
LC783           ldb a,u                         ; get number of creatures required
                beq LC78D                       ; brif none
LC787           jsr LCFA5                       ; create a creature
                decb                            ; created enough creatures?
                bne LC787                       ; brif not
LC78D           deca                            ; move on to next creature
                bpl LC783                       ; brif not finished all creatures
                ldu #creaturetab-17             ; point to creature table
                clr objiterstart                ; set to iterate from beginning of object table
LC795           jsr LCF63                       ; go fetch object
                beq LC7B6                       ; brif no more objects
                tst 5,x                         ; is object carried?
                bpl LC795                       ; brif so - fetch another
LC79E           leau 17,u                       ; move to next creature entry
                cmpu #mazedata                  ; are we at the end of the creature table?
                blo LC7AA                       ; brif not - use this creature
                ldu #creaturetab                ; point to start of creature table
LC7AA           tst 12,u                        ; is creature alive?
                beq LC79E                       ; brif not
                ldd 8,u                         ; get existing creature inventory
                stx 8,u                         ; put this object at start of creature inventory
                std ,x                          ; now put remaining inventory in the "next" pointer
                bra LC795                       ; go place another object
LC7B6           lda currentlevel                ; get current level
                anda #1                         ; set to "1" for odd, "0" for even
                nega                            ; negate - set to 00 for even, ff for odd
                sta levbgmask                   ; set level background mask
                sta commandarea+6               ; set background mask for command area
                sta infoarea+6                  ; set background mask for text area
                coma                            ; invert mask
                sta statusarea+6                ; set background mask for status line
                rts                             ; return to caller
; From here until the SWI routine jump table is the sound handling system. Any frequencies listed in the
; descriptions of these routines are for illustrative purposes as they are almost certainly wrong to a
; greater or lesser degree.
;
; swi 27 routine
; play a sound specified by the immediate identifier
LC7C8           ldx 12,s                        ; fetch return address
                lda ,x+                         ; fetch immediate data
                stx 12,s                        ; update return address
                ldb #$ff                        ; set to maximum volume
; swi 28 routine
; play a sound specified by the value in A
LC7D0           stb soundvol                    ; set the volume for the sound playing routine
                ldx #LC7DC                      ; point to sound routine jump table
                lsla                            ; two bytes per jump table entry
                jsr [a,x]                       ; call the sound generator routine
                clr PIA1                        ; turn off sound output
                rts                             ; return to caller
; the jump table for sound routines
LC7DC           fdb LC82B                       ; sound 0 - spider sound
                fdb LC850                       ; sound 1 - viper sound
                fdb LC951                       ; sound 2 - club giant sound
                fdb LC83C                       ; sound 3 - blob sound
                fdb LC8E2                       ; sound 4 - knight sound
                fdb LC955                       ; sound 5 - axe giant sound
                fdb LC84A                       ; sound 6 - scorpion sound
                fdb LC8DE                       ; sound 7 - shield knight sound
                fdb LC84D                       ; sound 8 - wraith sound
                fdb LC959                       ; sound 9 - galdrog sound
                fdb LC877                       ; sound 10 - wizard's image sound
                fdb LC877                       ; sound 11 - wizard sound
                fdb LC80A                       ; sound 12 - flask sound
                fdb LC811                       ; sound 13 - ring sound
                fdb LC827                       ; sound 14 - scroll sound
                fdb LC8DA                       ; sound 15 - shield sound
                fdb LC8A6                       ; sound 16 - sword sound
                fdb LC8B2                       ; sound 17 - torch sound
                fdb LC93F                       ; sound 18 - attack hit (player)
                fdb LC8E6                       ; sound 19 - attack hit (creature)
                fdb LC872                       ; sound 20 - walk into wall sound
                fdb LC86D                       ; sound 21 - creature death
                fdb LC88A                       ; sound 22 - wizard fade sound
; sound 12 - flask
LC80A           ldu #LC823                      ; point to 410Hz base tone
                lda #4                          ; repeat sound 4 times
                bra LC816                       ; go do the sound
; sound 13 - ring
LC811           ldu #LC81F                      ; point to 780Hz base tone
                lda #10                         ; repeat sound 10 times
LC816           sta soundrepeat                 ; set repeat counter
LC818           jsr ,u                          ; make a sound
                dec soundrepeat                 ; have we done enough of them?
                bne LC818                       ; brif not
                rts
; These routines produce a "sliding" tone starting at the base frequency. The specified base
; frequency is a rough estimate. The tones are created using square waves. After each full wave,
; the delay in reduced by one which increases the frequency. The last cycle is with the delay
; equal to 1 which yields an approximate frequency of 9520Hz. Because the delays become progressively
; shorter, the lower frequency range lasts longer than the higher frequency range.
;
```

```
; The "fcb $10" instructions turn the following LDX into LDY, effecting skipping them.
LC81F           ldx #$40                        ; set low frequency of sliding tone to ~782Hz
                fcb $10
LC823           ldx #$80                        ; set low frequency of sliding tone to ~413Hz
                fcb $10
; sound 14 - scroll
LC827           ldx #$100                       ; set low frequency of sliding tone to ~212HzHz
                fcb $10
; sound 0 - spider
LC82B           ldx #$20                        ; set low frequency of sliding tone to ~1416Hz
LC82E           bsr onesquarewave               ; do one square wave
                leax -1,x                       ; reduce delay (increase frequency)
                bne LC82E                       ; brif not yet reached maximum frequency
                rts
; Output a square wave with wave time defined by delay in X.
; The frequency of the wave is per the following table, which is calculated based on the the
; clock rate of 894886.25 cycles per second and the total time taken for this routine to
; execute. The total time for this routine to execut is 120+16X cycles where X is the value
; in X. So, the table is as follows. The X values are in hexadecimal. The frequency values
; are in decimal.
;
; X     Frequency
; 0001  6580Hz
; 0020  1416Hz
; 0040  782Hz
; 0080  413Hz
; 0100  212Hz
; FFFF  0.8533Hz
onesquarewave   lda #$ff                        ; (2~) hold DAC high for delay in X
                bsr setdacdel                   ; (7~)
                clra                            ; (2~) hold DAC low for delay in X
                bra setdacdel                   ; (3~)
; sound 3 - blob
; Output a series of 16 ascending tones with a base frequency descending from 43.4Hz to 27.2Hz.
LC83C           ldx #$500                       ; set for an ascending tone from 43.4Hz
LC83F           bsr onesquarewave               ; go make the sound
                leax $30,x                      ; decrease starting tone frequency by a bit
                cmpx #$800                      ; have we reached 27.2Hz?
                blo LC83F                       ; brif not
                rts
; sound 6 - scorpion
LC84A           lda #2                          ; two bits for scorpion
                skip2
; sound 8 - wraith
LC84D           lda #1                          ; one bit for wraith
                skip2
```

```
; sound 1 - viper
; This generates a sequence of sounds at notionally 5524Hz but it uses random amplituds so
; it's more of a random sound. The sound lasts about 35ms
LC850           lda #10                         ; ten bits for viper
                sta soundrepeat2                ; save repeat count
LC854           ldy #$c0                        ; number of iterations for tone generation
LC858           bsr sndseqnext                  ; (7~) get a sequence value
                bsr setdac                      ; (7~) set the dac
                leay -1,y                       ; (5~) done enough iterations?
                bne LC858                       ; (3~) brif not
                bsr LC8BA                       ; delay for 36.6 ms
                dec soundrepeat2                ; done repeats?
                bne LC854                       ; brif not
                rts                             ; return to caller
; This entry point takes a delay in X and programs the DAC with a value from the sequence generator.
; It exits after waiting out the X delay. It uses the MSB of the sequence value.
setdacseqdel    bsr sndseqnext                  ; (7~) get a value from the sequence to set the DAC
; This entry point takes a delay in X and the DAC value in A. It programs the DAC and waits out
; the delay in X.
setdacdel       bsr setdac                      ; (7~) program the DAC
                bra snddelay                    ; (3~) count down delay non-destructively
; sound 21 - creature death
; This does a slightly longer variation of the last sound for sound 22 below:
; A bust sliding from 622Hz to 162Hz, frequency shifting every 2.5 waves.
; This routine spins the sequence 640 times.
LC86D           ldu #LDBDA                      ; point to creature death tone generator parameters
                bra LC893                       ; go process the sound
; sound 20 - walk into wall
; This one uses exactly the same tone as the first half of sound 22.
; That is a short burst sliding from 405Hz to 162Hz, frequency shifting every half wave
; This routine spins the sequence 104 times.
LC872           ldu #LDBD2                      ; point to the generation specification for the sound
                bra LC893                       ; go generate the sound
; sound 10, sound 11 - wizard's image, wizard
LC877           lda #8                          ; do 8 iterations of this scheme
                sta soundrepeat                 ; set iteration counter
LC87B           bsr sndseqnext                  ; calculate new delay factor
                clra                            ; lose MSB
                lsrb                            ; double delay factor
                bne LC882                       ; brif not zero
                incb                            ; make sure don't do a massive delay
LC882           tfr d,x                         ; put delay into correct register
                bsr LC82E                       ; do a sliding tone
                dec soundrepeat                 ; have we done enough yet?
                bne LC87B                       ; brif not
; sound 22 - sound made just as a wizard fades out
; start with a short burst sliding from 405Hz to 162Hz, frequency shifting every half wave
; then, delay 36.6ms
; then, do a longer burst sliding from 622Hz to 162Hz, frequency shifting every two waves
; both bursts have semi-random amplitude derived from the sequence generator.
; For this sound, the sequence will be spun 616 times.
LC88A           ldu #LDBD2                      ; point to tone generator info
                bsr LC893                       ; process first pair
                bsr LC8BA                       ; delay for 36.6ms
                leau 4,u                        ; move to next pair of values
LC893           ldx ,u                          ; get delay value (frequency)
LC895           ldy 2,u                         ; get wave count for each frequency
LC898           bsr setdacseqdel                ; set the dac for the first half-wave
                leay -1,y                       ; are we done yet?
                bne LC898                       ; brif not
                leax 2,x                        ; lengthen delay slightly (reduce frequency)
                cmpx #$150                      ; are we at the minimum frequency (163Hz)?
                bne LC895                       ; brif not - get wave count again and keep going
                rts                             ; return to caller
; sound 16 - sword
; Uses random amplitude on an ascending volumn scale (roughly 510 iterations)
LC8A6           jsr LC931                       ;* set for ascending volume from 0 to $ff with a step of 0.5
                fcb $80                         ;*
LC8AA           bsr LC922                       ; apply step and program DAC
                bcs LC8B2                       ; brif counter wrapped
                bsr setdac                      ; set the DAC
                bra LC8AA                       ; keep looping
; sound 17 - torch
; Uses a random amplitude on a descending volume scale (roughly 405 iterations)
LC8B2           jsr LC92E                       ;* set for descending volume from $ffff with a step of 0.625
                fcb $a0                         ;*
LC8B6           bsr LC926                       ; apply step, multiplier, and set the dac - will return to our
caller when done
                bra LC8B6                       ; go apply another step
LC8BA           ldx #$1000                      ; delay factor for 36.6ms
; This routine counts X down nondestructively. It takes 16+8n cycles where
; n is the value in X.
snddelay        pshs x                          ; (7~) save delay counter
snddelay000     leax -1,x                       ; (5~) has timer expired?
                bne snddelay000                 ; (3~) brif not
LC8C3           puls x,pc                       ; (9~) restore delay counter and return
; This routine programs the DAC with the intensity in A adjusted by the sound volume.
; This routine takes 27 cycles.
setdac          ldb soundvol                    ; (5~) fetch volume multiplier for sound
                mul                             ; (11~) multiply it by the value we're trying to set
                anda #$fc                       ; (2~) lose the non-DAC bits
                sta PIA1                        ; (5~) set DAC
                rts                             ; (5~)
; This routine is a sequence generator with a period of 32768. soundseqseed is never initialized except
sndseqnext      ldd soundseqseed                ; (5~) fetch current value
                lslb                            ;* (2~) multiply by 4
                rola                            ;* (2~)
                lslb                            ;* (2~)
                rola                            ;* (2~)
                addd soundseqseed               ; (6~) add to previous value
                incb                            ; (2~) bump lsb
                std soundseqseed                ; (5~) save new value
                rts                             ; (5~) return to caller
; sound 15 - shield
; Run a dual wave with a low wave of 955Hz and a high wave of 3020Hz
LC8DA           bsr sndrundualwave
                fdb $6424
; sound 7 - shield knight
; Run a dual wave with a low wave of 1670Hz and a high wave of 3195Hz
LC8DE           bsr sndrundualwave
                fdb $3212
; sound 4 - knight
; Run a dual wave with a low wave of 580Hz and a high wave of 1575Hz
LC8E2           bsr sndrundualwave
                fdb $AF36
; sound 19 - attack hit against player
; Run a dual wave with a low wave of 2660Hz and a high wave of 4300Hz
LC8E6           bsr sndrundualwave
                fdb $1909
; This routine runs essentially a dual tone. The "frequency" of the lower bits is determined by the value
; in sndlowtonedel. The frequency of the high bit is determined by the delay in sndhitonedel. The two
frequencies run
; independently.
LC8EA           bsr LC92E                       ;* set up for descending volume with a step of 0.375
                fcb $60                         ;*
LC8ED           ldx sndlowtonedel               ; fetch low bits flip rate
                ldy sndhitonedel                ; fetch high bit flip rate
                clra                            ; initialize both "waves" to low
LC8F3           leax -1,x                       ; (5~) have we timed out on this level?
                bne LC8FD                       ; (3~) brif not
                ldx sndlowtonedel               ; (5~) reset counter
                eora #$7f                       ; (2~) flip all low bits of dac value
                bsr LC90A                       ; (7~) apply step and scale - will return to our caller when
things overflow (111~)
LC8FD           leay -1,y                       ; (5~) have run through the other sequence
                bne LC8F3                       ; (3~) brif not - start again
                ldy sndhitonedel                ; (6~) reset counter
                eora #$80                       ; (2~) flip high bit of dac value
                bsr LC90A                       ; (7~) apply step and scale - will return to our caller whent hings overflow (111~)
                bra LC8F3                       ; (3~) go check things again
LC90A           sta sndtemp                     ; (4~) save dac value
                bsr LC97E                       ; (7~) go calculate step and multiplier (53~)
                bls LC8C3                       ; (3~) skip the caller to this routine and return to its caller (PULS X,PC) if we wrapped
                bsr setdac                      ; (7~) set the dac (28~)
                lda sndtemp                     ; (4~) get back original dac value
                rts                             ; (5~) return to caller
; this routine doesn't return to the caller but to the caller's caller
sndrundualwave  ldx ,s++                        ; fetch location of parameters
                ldb ,x+                         ; fetch delay constant for low wave
                clra                            ; zero extend
                std sndlowtonedel               ; save it
                ldb ,x+                         ; fetch delay constant for high wave
                std sndhitonedel                ; save it
                bra LC8EA                       ; go process sound
LC922           bsr sndseqnext                  ; get a value from the sequence
                bra LC98D                       ; apply step and multplier (ascending)
LC926           bsr sndseqnext                  ; get value from sequence
LC928           bsr LC97E                       ; apply step and multiplier (descending)
                bls LC8C3                       ; skip the caller to this routine and return to its caller if we wrapped
                bra setdac                      ; set the dac and return
LC92E           ldx allones                     ; set initial base value to $ffff
                fcb $10                         ; go set up the step value
LC931           ldx zero                        ; set initial base value to $0000
LC933           stx sndampmult                  ; save initial base multiplier
                ldx ,s                          ; get return address
                ldb ,x+                         ; fetch step value
                clra                            ; zero extend
                std sndampstep                  ; save step value
                stx ,s                          ; save return address to be after step value
                rts                             ; return to caller
; sound 18 - attack hit against creature
; This is a sort of noisy square wave with a rough frequency of 4360Hz
LC93F           bsr LC92E                       ;* set up a countdown with step 0.375
                fcb $60                         ;*
LC942           jsr sndseqnext                  ; get a sequence value
                lsra                            ; make it in the low half of the range
                bsr LC928                       ; apply step and multiplier (descending) (will return to caller when overflow)
                jsr sndseqnext                  ; get another sequence value
                ora #$80                        ; force it high
                bsr LC928                       ; apply the step and multiplier (descending) (will return to our caller when overflow)
                bra LC942                       ; keep looping
; These three are basically the same sound. However, the stronger creatures have longer sounds that take longer
; to reach full volume, and thus longer to complete. The axe giant is roughly twice as long as the club giant and
; the galdrog is roughly three times as long as the club giant.
; sound 2 - club giant
LC951           ldx #$300                       ; step value for club giant (3)
                fcb $10
; sound 5 - axe giant                           ; step value for axe giant (2)
LC955           ldx #$200
                fcb $10
; sound 9 - galdrog                             ; step value for galdrog (1)
LC959           ldx #$100
                stx sndampstep                  ; save step value
                clra                            ; starting value at 0 (count up)
                clrb
                std sndampmult                  ; set starting multiplier
LC962           bsr LC922                       ; get a value from the sequence and apply multiplier
                bcs LC971                       ; brif we overflowed - done
                jsr setdac                      ; set the dac
                ldx #$f0                        ; delay for roughly 200Hz
                jsr snddelay                    ; go delay
                bra LC962                       ; go run another half wave
LC971           bsr LC92E                       ;* set up for a count down with a step of 0.25
                fcb $40                         ;*
LC974           bsr LC926                       ; get a sequence value and apply the step (descending), will return to our caller when done
                ldx #$60                        ; get delay roughly equal to  1050Hz
                jsr snddelay                    ; do the delay
                bra LC974                       ; go do another half wave
LC97E           pshs a                          ; (6~) save sequence value
                ldd sndampmult                  ; (5~) get mulitplier base
                subd sndampstep                 ; (6~) apply step value
LC984           pshs cc                         ; (6~) save result of subtraction
                std sndampmult                  ; (5~) save new multiplier base
                ldb 1,s                         ; (5~) get back dac value
                mul                             ; (11~) apply multiplier - use MSB in A
                puls cc,b,pc                    ; (9~) restore registers and return
LC98D           pshs a                          ; save sequence value
                ldd sndampmult                  ; get multiplier base
                addd sndampstep                 ; add step value
                bra LC984                       ; go deal with multiplier
; this is the swi routine offset table - each byte is the difference between the entry point
; of the previous routine and itself
LC995           fcb 0                           ; first routine has nothing before it
                fcb LC3A2-LC384
                fcb LC448-LC3A2
                fcb LC454-LC448
                fcb LC459-LC454
                fcb LC46F-LC459
                fcb LC472-LC46F
                fcb LC4CF-LC472
                fcb LC4F3-LC4CF
                fcb LC4F6-LC4F3
                fcb LC4FF-LC4F6
                fcb LC507-LC4FF
                fcb LC529-LC507
                fcb LC5D9-LC529
                fcb LC656-LC5D9
                fcb LC674-LC656
                fcb LC67F-LC674
                fcb LC686-LC67F
                fcb LC688-LC686
                fcb LC6A4-LC688
                fcb LC6A8-LC6A4
                fcb LC6C5-LC6A8
                fcb LC6E6-LC6C5
                fcb LC6FB-LC6E6
                fcb LC71F-LC6FB
                fcb LC743-LC71F
                fcb LC759-LC743
                fcb LC7C8-LC759
                fcb LC7D0-LC7C8
;*******************************************************************************************************************
; The following code handles displaying text on the screen. It works as follows.
;
; The graphics screen is divided into a grid of character cells 32 columns wide by 24 rows high. Each cell
; is 8 pixels wide by 8 pixels high. Text can be rendered anywhere on the screen as long as it fits within
; a character cell. The cells line up on even bytes which makes actually rendering the characters fast.
;
; Characters are encoded in 5 bits as follows: A through Z are given codes 1 through 26. 0 is a space. 27
; is the exclamation point, 28 is the underscore, 29 is the question mark, and 30 is the period. Code 31
; is used as a carriage return. Codes 32 and 33 are the left and right parts of the contracted heart symbol
; while 34 and 35 are the left and right parts of the expanded heart symbol. 36 is backspace.
;
; Glyphs for codes 0 through 30 are encoded using the packed five bit encoding and are located at LDB1B. They
; are encoded in a 5 by 7 bitmap which is shifted to be offset one pixel from the left of the character cell
; upon decoding.
;
; The glyphs for the heart codes are in unpacked encoding and are located at LDBB6 and occupy the entire
; 8 bit width of the character cell.
;
```

```
; These routines expect a pointer to the text configuration parameters in U. At offset 0 is the start address
; of the scrollable area of the screen (memory address). At offset 2 is the ending character cell address of
; the scrollable area of the screen. At offset 4 is the current printing position. At offset 6 is a mask with
; all pixels set to the background colour. At offset 7 a flag which when nonzero inhibits rendering text to
; the secondary graphics screen area. For the ordinary command entry area at the bottom of the screen, this
; will point to commandarea.
LC9B2           cmpa #$24                       ; is it backspace?
                beq LC9BF                       ; brif so
                cmpa #$1f                       ; vertical spacer?
                beq LC9CA                       ; brif so
                bsr LCA17                       ; go handle a glyph
                leax 1,x                        ; move to next character position
                rts                             ; return to caller
LC9BF           leax -1,x                       ; move display pointer back one
                cmpx allones                    ; did we wrap around negative?
                bne LC9C9                       ; brif not
                ldx 2,u                         ; get end of text area
                leax -1,x                       ; move back one position to be in the text area
LC9C9           rts                             ; return to caller
LC9CA           leax $20,x                      ; move pointer forward one character row
                exg d,x                         ; move pointer so we can do math
                andb #$e0                       ; force pointer to the start of the line
                exg d,x                         ; put pointer back where it belongs
                rts                             ; return to caller
LC9D4           pshs a,b,x,y                    ; save registers
                ldx ,u                          ; get start of screen address
                ldd 2,u                         ; get end of text area
                subd #$20                       ; knock one character row off it
                std 2,s                         ; save new display location
                bsr LCA10                       ; multiply by 8 - 8 pixel rows per cell
                tfr d,y                         ; save counter
LC9E3           ldd $100,x                      ; get bytes 8 pixel rows ahead
                tst 7,u                         ; do we need to skip the second screen?
                bne LC9EF                       ; brif so
                std $1800,x                     ; save scroll data on second screen
LC9EF           std ,x++                        ; save scroll data and move pointer ahead
                leay -2,y                       ; are we done yet?
                bne LC9E3                       ; brif not
                ldb 6,u                         ; fetch current background colour
                sex                             ; and make A match
                ldy #$100                       ; number of bytes to blank bottom row
LC9FC           tst 7,u                         ; are we doing second screen too?
                bne LCA04                       ; brif not
                std $1800,x                     ; blank pixels in second screen
LCA04           std ,x++                        ; blank pixels and move pointer forward
                leay -2,y                       ; are we done yet?
                bne LC9FC                       ; brif not
                puls a,b,x,y,pc                 ; restore registers and return
LCA0C           lslb                            ;* enter here to shift D left 5 bits
                rola                            ;*
                lslb                            ;*
                rola                            ;*
LCA10           lslb                            ;* enter here to shift D left 3 bits
                rola                            ;*
LCA12           lslb                            ;* enter here to shift D left 2 bits
                rola                            ;*
                lslb                            ;*
                rola                            ;*
                rts
LCA17           pshs a,b,x,y,u                  ; save registers
                cmpa #$20                       ; is it a printing character?
                blo LCA29                       ; brif so
                suba #$20                       ; mask off printing characters
                ldb #7                          ; 7 bytes per font table entry
                mul                             ; get offset in table
                addd #LDBB6                     ; add in base address of table
                tfr d,x                         ; put font pointer somewhere useful
                bra LCA44                       ; go draw glyph
LCA29           ldb #5                          ; 5 bytes per font table entry
                mul                             ; get offset in table
                addd #LDB1B                     ; add in base address of table
                tfr d,x                         ; put pointer somewhere useful
                ldu #fontbuf                    ; point to buffer to decode glyph data
                decodestr                       ; go decode a packed string
                ldx #fontbuf+7                  ; point one past end of buffer
LCA39           lsl ,-x                         ;* centre glyph data in byte
                lsl ,x                          ;*
                cmpx #fontbuf                   ; at start of buffer?
                bhi LCA39                       ; brif not - keep centring
                ldu 6,s                         ; get back U value
LCA44           ldd 4,u                         ; get display address location
                bsr LCA10                       ; multiply by 8 - gets start of row in 11..8
                lsrb                            ;* and divide lsb by 8 again to get offset within
                lsrb                            ;* the row to bits 4..0
                lsrb                            ;* and force to top of character cell
                addd ,u                         ; add in start of text area
                tfr d,y                         ; put pointer somewhere useful
                ldb #7                          ; seven bytes to copy
LCA51           lda ,x+                         ; get byte from font data
                eora 6,u                        ; merge with background colour
                sta ,y                          ; save it on the screen
                tst 7,u                         ; do we need to update second screen?
                bne LCA5F                       ; brif not
                sta $1800,y                     ; save pixels on second screen
LCA5F           leay $20,y                      ; move display pointer down one pixel row
                decb                            ; are we done yet?
                bne LCA51                       ; brif not - do another
                puls a,b,x,y,u,pc               ; restore registers and return
; This routine divides a 16 bit unsigned value in D by a 16 bit unsigned value in X. The result
; will be in D with the binary point to the right of A.
LCA67           pshs a,b,x                      ; make hole for result and save divisor
                clr ,s                          ;* initialize quotient
                clr 1,s                         ;*
                clr accum0                      ; use accum0 for extra precision on dividend
                std accum0+1                    ; save dividend
                beq LCA97                       ; brif dividend is zero - nothing to do
                cmpd 2,s                        ; is dividend equal to divisor?
                bne LCA7C                       ; brif not
                inc ,s                          ; set quotient to 1.0
                bra LCA97                       ; go return
LCA7C           ldx #16                         ; we need to do 16 iterations
LCA7F           lsl accum0+2                    ;* shift dividend
                rol accum0+1                    ;*
                rol accum0                      ;*
                lsl 1,s                         ;= shift quotient
                rol ,s                          ;=
                ldd accum0                      ; get dividend high word
                subd 2,s                        ; subtract out divisor
                bcs LCA93                       ; brif it doesn't go
                std accum0                      ; save new dividend residue
                inc 1,s                         ; record the fact that it went
LCA93           leax -1,x                       ; have we done all 16 bits?
                bne LCA7F                       ; brif not
LCA97           puls a,b,x,pc                   ; fetch result, restore registers, and return
LCA99           coma                            ;* do a one's complement of D
                comb                            ;*
                addd #1                         ; adding 1 turns it into negation
                rts                             ; return to caller
LCA9F           pshs a,b,x                      ; save registers
                ldx pixelcount                  ; get number of pixels to draw
                ldd ,s                          ; get the difference
                bpl LCAAE                       ; brif positive
                bsr LCA99                       ; negate difference
                bsr LCA67                       ; divide by number of pixels
                bsr LCA99                       ; negate the result
                skip2                           ; skip next instruction
LCAAE           bsr LCA67                       ; divide by number of pixels
                std ,s                          ; save step value
                puls a,b,x,pc                   ; restore registers and return
LCAB4           jmp LCB8A                       ; go return from the line drawing routine
; Draw a line from (xbeg,ybeg) to (xend,yend) respecting the light level in the dungeon (lightlevel)
; which is used as a step count between when to actually draw pixels.
;
; Variables used:
; lightlevel    the current light level in the dungeon
; lightcount    how many pixels left before we actually draw another
; ybeg          input start Y
; xbeg          input start X
; yend          input end Y
; xend          input end X
; xcur          X coordinate of pixel to be drawn (24 bits with 8 bits after binary point)
; ycur          U cpprdomate of pixel to be drawn (24 bits with 8 bits after binary point)
; xpstep        24 bit X coordinate difference (per pixel)
; ypstep        24 bit Y coordinate difference (per pixel)
; pixelcount    number of pixels to draw in the line
; xbstep        the offset for when X coordinate goes to a new byte
; xystep        the offset for when Y coordinate goes to a new line
; drawstart     the start address of the graphics screen area the line is within
; drawend       the end address of the graphics screen area the line is within
; accum0                a temporary scratch variable
;
; Note: ypstep+1 and xpstep+1 are also used as temporary holding values for the
; integer difference in the Y and X coordinates respectively.
drawline        pshs a,b,x,y,u                  ; save registers
                inc lightlevel                  ; are we completely dark?
                beq LCAB4                       ; brif so - we can short circuit drawing entirely
                lda lightlevel                  ; get light level in dungeon
                sta lightcount                  ; save in working count (skip count for pixel drawing)
                ldd xend                        ; get end X coordinate
                subd xbeg                       ; subtract start X coordinate
                std xpstep+1                    ; save coordinate difference
                bpl LCACB                       ; brif positive difference
                bsr LCA99                       ; negate the difference
LCACB           std pixelcount                  ; save absolute value of X difference as pixel count
                ldd yend                        ; get end Y coordinate
                subd ybeg                       ; subtract start Y coordinate
                std ypstep+1                    ; save coordinate difference
                bpl LCAD7                       ; brif positive difference
                bsr LCA99                       ; negate the difference
LCAD7           cmpd pixelcount                 ; is the Y difference bigger than X?
                blt LCAE0                       ; brif not
                std pixelcount                  ; save Y difference as pixel count
                beq LCAB4                       ; brif no pixels to draw - short circuit
LCAE0           ldd xpstep+1                    ; get X difference
```

```
                bsr LCA9F                       ; calculate X stepping value
                std xpstep+1                    ; save X stepping value
                tfr a,b                         ; save msb of difference
                sex                             ; sign extend it
                ldb #1                          ; X stepping value - 1 for ascending
                sta xpstep                      ; sign extend stepping difference to 24 bits
                bpl LCAF0                       ; brif positive
                negb                            ; set stepping value to -1
LCAF0           stb xbstep                      ; save X byte stepping value
                ldd ypstep+1                    ; get Y difference
                bsr LCA9F                       ; calculate Y step value
                std ypstep+1                    ; save result
                tfr a,b                         ; save msb of difference
                sex                             ; sign extend it
                ldb #$20                        ; Y byte stepping value - 32 bytes per line, ascending
                sta ypstep                      ; sign extend the difference to 24 bits
                bpl LCB02                       ; brif positive
                negb                            ; negate the difference - -32 bytes per line, descending
LCB02           stb xystep                      ; save Y byte stepping value
                ldd xbeg                        ; get start X coordinate
                std xcur                        ; save in X coordinate counter
                ldd ybeg                        ; get start Y coordinate
                std ycur                        ; save in Y coordinate counter
                lda #$80                        ; value for low 8 bits to make the values ".5"
                sta xcur+2                      ; set X coordinate to ".5"
                sta ycur+2                      ; set Y coordinate to ".5"
                ldx 2,u                         ; get end of graphics area address
                stx drawend                     ; save it for later
                ldx ,u                          ; get start of graphics area address
                stx drawstart                   ; save it for later
                ldd ycur                        ; get Y coordinate for pixel
                jsr LCA0C                       ; shift left 5 bits - 32 bytes per row
                leax d,x                        ; add to screen start address
                ldd xcur                        ; get X coordinate for pixel
                jsr asrd3                       ; shift right 3 bits - 8 pixels per byte
                leax d,x                        ; add to row start address
                ldu #LCB8E                      ; point to table of pixel masks
                ldy pixelcount                  ; get number of pixels to draw
LCB2E           dec lightcount                  ; are we ready to draw another pixel (due to light level)?
                bne LCB54                       ; brif not
                lda lightlevel                  ; get light level
                sta lightcount                  ; reset current "pixel delay"
                tst xcur                        ; is X coordinate off the right of the screen?
                bne LCB54                       ; brif so
                cmpx drawstart                  ; is the pixel address before the start of the graphics area?
                blo LCB54                       ; brif so
                cmpx drawend                ; is the pixel address after the end of the graphics area?
                bhs LCB54                   ; brif so
                ldb xcur+1                  ; get X coordinate lsb
                andb #7                     ; mask off low 3 bits for offset in byte
                lda b,u                     ; get pixel mask to use
                tst levbgmask               ; currently using black background?
                beq LCB50                   ; brif so
                coma                        ; invert mask for white background
                anda ,x                     ; merge in existing graphics data
                skip2                       ; skip next instruction
LCB50           ora ,x                      ; merge in existing graphics data (black background)
                sta ,x                      ; save new graphics data on the screen
LCB54           lda xcur+1                  ; get X coordinate lsb
                anda #$f8                   ; mask off the pixel offset in the byte
                sta accum0                  ; save it for later
                ldd xcur+1                  ; get X coordinate low bits
                addd xpstep+1               ; add in X difference
                std xcur+1                  ; save new low bits for X coordinate
                ldb xcur                    ; get X coordinate high bits
                adcb xpstep                 ; add in difference high bits
                stb xcur                    ; save new X coordinate high bits
                anda #$f8                   ; mask off pixel offset in data byte
                cmpa accum0                 ; are we in the same byte?
                beq LCB70                   ; brif so
                ldb xbstep                  ; get byte X step value
                leax b,x                    ; move pointer appropriately
LCB70           ldd ycur+1                  ; get Y coord low bits
                sta accum0                  ; save screen Y coordinate
                addd ypstep+1               ; add in Y step value low bits
                std ycur+1                  ; save new low bits
                ldb ycur                    ; get Y coord high bits
                adcb ypstep                 ; add in Y step value high bits
                stb ycur                    ; save new Y coord high bits
                cmpa accum0                 ; are we on the same scren row?
                beq LCB86                   ; brif so
                ldb xystep                  ; get Y byte step value
                leax b,x                    ; move pointer appropriately
LCB86           leay -1,y                   ; have we drawn all the pixels?
                bne LCB2E                   ; brif not - draw another
LCB8A           dec lightlevel              ; compensate for "inc" above
                puls a,b,x,y,u,pc           ; restore registers and return
LCB8E           fcb $80,$40,$20,$10         ; pixels 0, 1, 2, 3 (left to right) in byte
                fcb $08,$04,$02,$01         ; pixels 4, 5, 6, 7 (left to right) in byte
LCB96           pshs a,x,u                  ; save registers
                ldx linebuffptr             ; get input buffer/line pointer
                ldu #wordbuff               ; point to word buffer
```

```
LCB9D           lda ,x+                         ; get character from input
                beq LCB9D                       ; brif end of line
                bra LCBA5                       ; get on with things
LCBA3           lda ,x+                         ; get new character from input
LCBA5           ble LCBAF                       ; brif not valid character
                sta ,u+                         ; save filename character
                cmpu #wordbuffend               ; are we at the end of the buffer?
                blo LCBA3                       ; brif not - check another
LCBAF           lda #$ff                        ; put end of word marker
                sta ,u+
                stx linebuffptr                 ; save new input pointer location
                tst wordbuff                    ; set flags for whether we have a word
                puls a,x,u,pc                   ; restore registers and return
; Parse an object from command line
parseobj        clr parsegenobj                 ; flag generic object type
                ldx #kwlist_obj                 ; point to object type list
                bsr LCBEC                       ; look up word in object type list
                bmi parseobj000                 ; brif no match - try matching specific type
                beq badcommandret               ; brif no match - error out
                std parseobjtype                ; save object type matched
                rts                             ; return to caller
parseobj000     dec parsegenobj                 ; flag specific object type found
                ldx #kwlist_adj                 ; point to specific object types
                bsr LCBE7                       ; look up word in object type list
                ble badcommandret               ; brif no match
                std parseobjtype                ; save object type
                ldx #kwlist_obj                 ; point to generic object types
                bsr LCBEC                       ; look up keyword
                ble badcommandret               ; brif no match
                cmpb parseobjtypegen            ; did the object type match?
                bne badcommandret               ; brif not
                rts                             ; return to caller
badcommandret   leas 2,s                        ; don't return to caller - we're bailing out
badcommand      renderstrimmp                   ; display "???" for unknown command
                fcb $17,$7b,$d0                 ; packed "???" string
                rts                             ; return to caller's caller
LCBE7           pshs a,b,x,y,u                  ; save registers
                clra                            ; initialize specific type to zero
                bra LCBF4                       ; go look up keyword
LCBEC           pshs a,b,x,y,u                  ; save registers
                clra                            ; initialize specific type to zero
                clrb                            ; initialize generic type to zero
                bsr LCB96                       ; parse a word from the input line
                bmi LCC2D                       ; brif no word present
LCBF4           clr kwmatch                     ; flag no match
                clr kwexact                     ; flag incomplete match
                ldb ,x+                         ; fetch number of keywords in list
                stb kwcount                     ; save it in temp counter
LCBFC           ldu #wordbuff                   ; point to decode buffer
                decodestrsb                     ; decode the keyword string
                ldy #stringbuf+1                ; point to decoded keyword string (past the object code)
LCC05           ldb ,u+                         ; get a character from word string
                bmi LCC17                       ; brif end of string
                cmpb ,y+                        ; does it match?
                bne LCC22                       ; brif not
                tst ,y                          ; are we at the end of the keyword?
                bpl LCC05                       ; brif not
                tst ,u                          ; are we at the end of the word?
                bpl LCC22                       ; brif not
LCC15           dec kwexact                     ; flag complete match
LCC17           tst kwmatch                     ; do we already have a match?
                bne LCC2B                       ; brif so
                inc kwmatch                     ; mark match found
                ldb stringbuf                   ; get the keyword code
                std ,s                          ; save keyword number and object code
LCC22           inca                            ; bump keyword count
                dec kwcount                     ; have we reached the end of the list?
                bne LCBFC                       ; brif not - check another keyword
                tst kwmatch                     ; do we have a match?
                bne LCC2F                       ; brif so
LCC2B           ldd allones                     ; flag error (-1)
LCC2D           std ,s                          ; save result
LCC2F           puls a,b,x,y,u,pc               ; restore registers and return value, return
LCC31           ldx #kwlist_dir                 ; point to direction keywords
                bsr LCBEC                       ; evaluate the specified keyword
                ble badcommandret               ; brif no matching keyword
                ldu #righthand                  ; point to right hand contents
                cmpa #1                         ; is it right hand wanted?
                beq LCC46                       ; brif so - return pointer
                ldu #lefthand                   ; point to left hand contents
                cmpa #0                         ; is it left hand wanted?
                bne badcommandret               ; brif not - error
LCC46           ldx ,u                          ; fetch object pointer to X (and set Z if nothing)
                rts
LCC49           pshs a,b,x,u                    ; save coordinates and registers
                deca                            ; look at rooms to the NE, N, NW
                bsr LCC56
                inca                            ; look at rooms to the E, W, <here>
                bsr LCC56
                inca                            ; look at rooms to the SE, S, SW
                bsr LCC56
                puls a,b,x,u,pc                 ; restore registers and return
```

```
LCC56           pshs a,b                        ; save coordinates
                decb                            ; look at room to W
                bsr LCC60
                incb                            ; look at room <here>
                bsr LCC60
                incb                            ; look at room E
                skip2                           ; skip next instruction
LCC60           pshs a,b                        ; save coordinates
                bsr LCC8E                       ; did we fall off side of map?
                bne LCC6B                       ; brif so
                bsr LCC7B                       ; get pointer to room data
                lda ,x                          ; fetch room data
                skip2                           ; skip instruction
LCC6B           lda #$ff                        ; flag no tunnel
                sta ,u+                         ; save data for this room
                puls a,b,pc                     ; save registers and return
LCC71           getrandom                       ; get a random number
                anda #$1f                       ; convert it to 0-31
                tfr a,b                         ; save it
                getrandom                       ; get another random number
                anda #$1f                       ; also convert it to 0-31
LCC7B           pshs a,b                        ; save coordinates
                anda #$1f                       ; force coordinates to range 0-31
                andb #$1f
                tfr d,x                         ; save coordinates for later
                ldb #32                         ; 32 rooms per row
                mul                             ; calculate row offset
                addd #mazedata                  ; convert to absolute pointer
                exg d,x                         ; get pointer to X, get back coordinates
                abx                             ; add offset within row
                puls a,b,pc                     ; restore coordinates and return pointer in X
LCC8E           pshs a,b                        ; save coordinates
                anda #$1f                       ; modulo the Y coordinate
                cmpa ,s                         ; does it match?
                bne LCC9A                       ; brif not - fell off side
                andb #$1f                       ; modulo the X coordinate
                cmpb 1,s                        ; does it match? (set flags)
LCC9A           puls a,b,pc                     ; return Z set if not falling off side of map
; This routine creates a maze for the specified level number.
createmaze      ldx #mazedata                   ; get start address to set to $ff
                ldu #mazedata+1024              ; get end address
                setblock                        ; go set block to $ff
                ldx #levelseeds                 ; point to level seeds table
                ldb currentlevel                ; fetch current level
                abx                             ; offset into table (the seeds overlap!)
                ldd ,x++                        ; fetch first two bytes of level seed
                std randomseed                  ; set random seed
                lda ,x                          ; fetch third byte of level seed
                sta randomseed+2                ; set random seed
                ldy #500                        ; dig out 500 rooms
                jsr LCC71                       ; fetch a random starting point
                std temploc                     ; save starting pointer
LCCBB           getrandom                       ; get random number
                anda #3                         ; select only 4 directions
                sta curdir                      ; save direction we're going
                getrandom                       ; get random number
                anda #7                         ; convert to value from 1-8
                inca
                sta genpathlen                  ; save number of steps to dig out
                bra LCCD2                       ; go dig the tunnel
LCCCA           ldd gencurcoord                 ; get current coordinate
                std temploc                     ; save it as starting position
                dec genpathlen                  ; have we gone far enough?
                beq LCCBB                       ; brif so - select a new direction
LCCD2           ldd temploc                     ; fetch maze coordinates
                jsr LD11B                       ; apply direction to coordinates
                bsr LCC8E                       ; did we fall off the side of the map?
                bne LCCBB                       ; brif so - select a new direction
                std gencurcoord                 ; save new coordinate
                tst ,x                          ; is this room open?
                beq LCCCA                       ; brif so - move to next
                ldu #neighbourbuff              ; point to temporary storage area
                jsr LCC49                       ; set bytes to FF or 00 depending on whether the rooms in the
3x3 area are open
                lda 3,u                         ; get W room
                adda ,u                         ; add data for NW room
                adda 1,u                        ; add data for N room
                beq LCCBB                       ; brif all open - get new direction
                lda 1,u                         ; get data for N room
                adda 2,u                        ; add data for NE room
                adda 5,u                        ; add data for E room
                beq LCCBB                       ; brif all open - get new direction
                lda 5,u                         ; get data for E room
                adda 8,u                        ; add data for SE room
                adda 7,u                        ; add data for S room
                beq LCCBB                       ; brif all open - get new direction
                lda 7,u                         ; get data for S room
                adda 6,u                        ; add data for SW room
                adda 3,u                        ; add data for W room
                beq LCCBB                       ; brif all open - get new direction
                clr ,x                          ; mark this room open
                leay -1,y                       ; have we dug out enough rooms?
                bne LCCCA                       ; brif not - keep digging
                clr temploc                     ; set coordinates to top left
                clr temploc+1
LCD11           ldd temploc                     ; get current coordinates
                jsr LCC7B                       ; convert to pointer
                lda ,x                          ; get room data
                inca                            ; is ot open?
                beq LCD41                       ; brif not
                ldd temploc                     ; get coordinates
                ldu #neighbourbuff              ; point to temp area
                jsr LCC49                       ; calculate neighbors
                lda ,x                          ; get room data at current room
                ldb #$ff                        ; data for "no room"
                cmpb 1,u                        ; is there a room N?
                bne LCD2D                       ; brif so
                ora #3                          ; flag as no exit N
LCD2D           cmpb 3,u                        ; is there a room W?
                bne LCD33                       ; brif so
                ora #$c0                        ; flag as no exit W
LCD33           cmpb 5,u                        ; is there a room E
                bne LCD39                       ; brif so
                ora #$0c                        ; flag as no exit E
LCD39           cmpb 7,u                        ; is there a room S?
                bne LCD3F                       ; brif so
                ora #$30                        ; flag as no exit S
LCD3F           sta ,x                          ; save adjusted room data
LCD41           ldb #32                         ; 32 rooms per row
                inc temploc+1                   ; bump X coordinate
                cmpb temploc+1                  ; did we wrap?
                bne LCD11                       ; brif not
                clr temploc+1                   ; reset to left edge
                inc temploc                     ; bump Y coordinate
                cmpb temploc                    ; did we wrap?
                bne LCD11                       ; brif not - fix another room's exits
                ldb #70                         ; create 70 doors
                ldu #doormasks                  ; pointer to routine to make a normal door
LC056           bsr LCD6D                       ; go create a door
                decb                            ; are we done yet?
                bne LC056                       ; brif not
                ldb #$2d                        ; create 45 magic doors
                ldu #mdoormasks                 ; pointer to routine to make a magic door
LCD60           bsr LCD6D                       ; go create a door
                decb                            ; done yet?
                bne LCD60                       ; brif not
                ldb clockctrs+2                 ; get number of times to spin the random number generator
(cycles once/minute)
```

```
LCD67           getrandom                       ; fetch a random number
                decb                            ; have we done enough randoms?
                bne LCD67                       ; brif not, do another
                rts                             ; return to caller
LCD6D           pshs a,b,x,y,u                  ; save registers
                ldy #dirmasks                   ; point to direction masks
LCD73           jsr LCC71                       ; get a random location
                std gencurcoord                 ; save coordinates
                ldb ,x                          ; get room data at location
                cmpb #$ff                       ; is there a room?
                beq LCD73                       ; brif not - try again
                getrandom                       ; get random number
                anda #3                         ; normalize to direction
                sta curdir                      ; save direction
                bitb a,y                        ; is there a door or wall at that direction?
                bne LCD73                       ; brif so - try again
                orb a,u                         ; mark the direction as having a door of desired type
                stb ,x                          ; save new room data
                ldd gencurcoord                 ; get back coordinates
                jsr LD11B                       ; get pointer to neighbor
                ldb curdir                      ; get direction back
                addb #2                         ; calculate opposite direction
                andb #3
                lda ,x                          ; get data at neighboring room
                ora b,u                         ; set it to the right type of door
                sta ,x                          ; save new neighbor data
                puls a,b,x,y,u,pc               ; restore data and return
; These are the random seeds for the level mazes. Note that the seeds overlap by two
; bytes. The actual seed values are:
; Level 1: 73c75d
; Level 2: c75d97
; Level 3: 5d97f3
; Level 4: 97f313
; Level 5: f31387
levelseeds      fcb $73,$c7,$5d,$97,$f3,$13,$87
dirmasks        fcb $03,$0c,$30,$c0             ; direction masks
doormasks       fcb $01,$04,$10,$40             ; direction masks to create doors
mdoormasks      fcb $02,$08,$20,$80             ; direction masks to create magic doors
; This routine draws the display for a scroll.
;
; If showseer is set to nonzero, it displays creature and object information (SEER SCROLL)
; otherwise it shows only the maze, holes, and player location (VISION SCROLL).
;
; temploc is used as a temporary scratch counter for displaying the maze itself.
displayscroll   ldu screendraw                  ; point to screen we're using to draw on
                ldd #$1f1f                      ; maximum X and Y coordinates
                std temploc                     ; save coordinates
LCDB9           ldd temploc                     ; fetch current coordinates
                bsr LCE11                       ; calculate absolute pointer to screen location
                jsr LCC7B                       ; fetch pointer to room data
                clrb                            ; initialize to black
                lda ,x                          ; fetch room data
                inca                            ; is it an empty room?
                bne LCDC7                       ; brif not
                decb                            ; set to white
LCDC7           lda #6                          ; set 6 rows
LCDC9           stb ,y                          ; set a row
                leay $20,y                      ; move to next row
                deca                            ; done all rows?
                bne LCDC9                       ; brif not
                dec temploc+1                   ; move left one space
                bpl LCDB9                       ; brif not at left yet
                lda #$1f                        ; max right coord
                sta temploc+1                   ; reset X coordinate to far right
                dec temploc                     ; move back a row
                bpl LCDB9                       ; brif still in map
                tst showseer                    ; are we showing creatures and objects?
                beq LCE2B                       ; brif not
                clr objiterstart                ; start iteration from scratch
LCDE3           jsr LCF63                       ; go fetch object
                beq LCDF7                       ; brif no more objects
                tst 5,x                         ; is the object equipped/carried?
                bne LCDE3                       ; brif so
                ldd 2,x                         ; get coordinates of object
                bsr LCE11                       ; get absolute address of location
                ldd #8                          ; object location symbol
                bsr LCE1D                       ; display symbol
                bra LCDE3                       ; go check another object
LCDF7           ldx #creaturetab-17             ; point to creature table
LCDFA           leax $11,x                      ; move to next creature
                cmpx #mazedata                  ; are we at the end of the creature table?
                beq LCE2B                       ; brif so
                tst 12,x                        ; is creature alive?
                beq LCDFA                       ; brif not
                ldd 15,x                        ; get current creature location
                bsr LCE11                       ; turn location into pointer
                ldd #$1054                      ; symbol for creature
                bsr LCE1D                       ; go display symbol
                bra LCDFA                       ; go check another creature
LCE11           tfr d,y                         ; save requested coordinates
                ldb #$c0                        ; calculate row offset based on display height of 6 px
                mul                             ; now we have the offset from the start of the screen
                addd ,u                         ; now D has the absolute address of the start of the line
                exg d,y                         ; put pointer in Y and get back coordinates
                leay b,y                        ; offset in the X direction for real pointer
                rts                             ; return to caller
LCE1D           sta $20,y                       ; set top row of symbol
                stb $40,y                       ; set second row of symbol
                stb $60,y                       ; set third row of symbol
                sta $80,y                       ; set bottom row of symbol
LCE2A           rts                             ; return to caller
LCE2B           ldd playerloc                   ; get current player position
                bsr LCE11                       ; calculate absolute address
                ldd #$2418                      ; bit patterns to create a *
                bsr LCE1D                       ; go mark the player position
                ldx holetabptr                  ; point to the hole table for this level
                bsr LCE38                       ; go display holes going up then fall through for holes going
down
LCE38           lda ,x+                         ; get hole type flag
                bmi LCE2A                       ; brif end of this table (return)
                ldd ,x++                        ; get coordinates
                bsr LCE11                       ; calculate absolute address
                ldd #$3c24                      ; symbol for displaying a hole
                bsr LCE1D                       ; go display symbol
                bra LCE38                       ; go check another entry
LCE47           pshs a,x                        ; save registers
                ldx #LCF48                      ; point to lighting level constants
                tst movehalf                    ; is this a half step render?
                bne LCE5C                       ; brif not
                leax >1,x                       ; move ahead in the render scale constants

                tst movebackhalf                ; is it a half step back?
LCE56           bne LCE5C                       ; brif not
                leax >-11,x                     ; move to backstep levels
LCE5C           lda renderdist                  ; get distance to render
                lda a,x                         ; get scale factor for the distance
                sta horizscale                  ; save horizontal scaling factor
                sta vertscale                   ; save vertical scaling factor
                puls a,x,pc                     ; restore registers and return
; This is the routine that shows the regular dungeon view.
LCE66           cleargfx2                       ; clear the graphics area
                clr renderdist                  ; set render distance to immediate
                ldd playerloc                   ; get player location
                std temploc                     ; save current render location
LCE6E           bsr LCE47                       ; calculate scaling factor for current render location
                ldd temploc                     ; fetch render location
                jsr LCC7B                       ; get maze pointer
                lda ,x                          ; get maze data for current location
                ldu #neighbourbuff              ; point to neighbor calculation buffer
                ldx #4                          ; check four directions
LCE7D           tfr a,b                         ; save door info
                andb #3                         ; check low 2 bits
                stb 4,u                         ;= save twice so we can handle rotation without special cases
                stb ,u+                         ;=
                lsra                            ;* shift room data to next direction
                lsra                            ;*
                leax -1,x                       ; have we done all four directions?
                bne LCE7D                       ; brif not
                ldb facing                      ; get the direction we're facing
                ldu #neighbourbuff              ; point to neighbor table
                leau b,u                        ; offset neighbor table
                ldy #LDBDE                      ; point to direction rendering table (pointers to graphic
elements)
LCE96           lda ,y+                         ; get table entry flag/direction
                bmi LCED8                       ; brif end of table
                ldb a,u                         ; get direction data
                lslb                            ; two bytes per door type
                cmpb #4                         ; is it a magic door?
                bne LCEA9                       ; brif not
                ldx b,y                         ; fetch graphic pointer
                dec rendermagic                 ; flag to render to magic light
                bsr LCECE                       ; go draw the magic door
                ldb #6                          ; change type to wall (invisible magic door)
LCEA9           ldx b,y                         ; get graphic
                bsr LCECE                       ; draw the graphic
                leay 8,y                        ; move to next table entry
                bra LCE96                       ; go handle another direction

LCEB1           rts                             ; return to caller
LCEB2           tfr x,y                         ; save graphic pointer
                tst b,u                         ; is there a door in that direction?
                bne LCEB1                       ; brif so
                addb facing                     ; calculate real direction
                stb curdir                      ; save real direction
                ldd temploc                     ; fetch render location
                jsr LD11B                       ; get new coordinates and room pointer
                jsr LCF82                       ; get creature in room
                beq LCEB1                       ; brif no creature in room
                exg x,y                         ; save creature pointer in Y, get original graphic pointer
back
LCEC8           tst 2,y                         ; is creature magical?
                beq LCECE                       ; brif not - use physical ight
                dec rendermagic                 ; render magic light
LCECE           pshs u                          ; save registers
                setlighting                     ; set light level
                ldu screendraw                  ; point to drawing screen
                drawgraphic                     ; draw the selected graphic
                puls u,pc                       ; restore registers and return
LCED8           ldd temploc                     ; get render location
                jsr LCF82                       ; get creature in room
                beq LCEEB                       ; brif no creature
                tfr x,y                         ; save creature pointer
                ldb 13,y                        ; get creature tpe
                lslb                            ; double it
                ldx #LDAA3                      ; point to creature graphics table
                ldx b,x                         ; get graphic data
                bsr LCEC8                       ; go render graphic
LCEEB           ldb #3                          ; right hand
                ldx #LDCB0                      ; point to graphic
                bsr LCEB2                       ; go render graphic if there's a door
                ldb #1                          ; left hand
                ldx #LDCB9                      ; point to graphic
                bsr LCEB2                       ; go render graphic if there's a door
                ldx #LDD3C                      ; point to graphic
                ldd temploc                     ; get current location
                jsr LCFE1                       ; get hole information
                bmi LCF09                       ; brif no hole
                ldx #LDCC2                      ; point to graphic table for holes
                lsla                            ; two bytes per pointer entry
                ldx a,x                         ; get actual graphic for the hole present
LCF09           bsr LCECE                       ; go render the graphic
                clr objiterstart                ; reset object iterator
LCF0D           ldd temploc                     ; get current room
                jsr LCF53                       ; fetch next object on floor
                beq LCF24                       ; brif no more objects
                lda 10,x                        ; get object type
                lsla                            ; double it - two bytes per pointer entry
                ldx #LD9EE                      ; point to object images
                ldx a,x                         ; get correct graphic image
                dec rendermagic                 ; set to render magic light
                bsr LCECE                       ; render object in magic light (why??)
                bsr LCECE                       ; render object in physical light
                bra LCF0D                       ; go handle another object in the room
LCF24           tst ,u                          ; any door looking forward?
                bne LCF3D                       ; brif so
                lda facing                      ; get direction facing
                sta curdir                      ; save direction going
                ldd temploc                     ; get current direction
                jsr LD11B                       ; get pointer in that direction
                std temploc                     ; save new location
                inc renderdist                  ; bump render distance (next room going forward)
                lda renderdist                  ; get distance
                cmpa #9                         ; is it 9 steps out?
                lble LCE6E                      ; brif 9 or less - render another room
LCF3D           rts                             ; return to caller
; These are the scale factors used for rendering rooms.
                fcb $c8,$80,$50,$32,$1f,$14,$0c,$08,$04,$02
LCF48           fcb $ff,$9c,$64,$41,$28,$1a,$10,$0a,$06,$03,$01
LCF53           bsr LCF63                       ; fetch next object in iteration
                beq LCF62                       ; brif no object
                cmpd 2,x                        ; is object at desired location
                bne LCF53                       ; brif not - try again
                tst 5,x                         ; is object in inventory?
                bne LCF53                       ; brif so - not in room
                andcc #$fb                      ; clear Z for found
LCF62           rts                             ; return to caller
LCF63           pshs a                          ; save register
                lda currentlevel                ; fetch current level
                ldx objiterptr                  ; fetch object pointer
                tst objiterstart                ; are we starting at beginning?
                bne LCF72                       ; brif not
                ldx #objecttab-14               ; point to start of table
                dec objiterstart                ; mark not at beginning any more
LCF72           leax 14,x                       ; move to next object
                stx objiterptr                  ; save object pointer for iteration
                cmpx objectfree                 ; are we at the end of the table?
                beq LCF80                       ; brif so - return
                cmpa 4,x                        ; is the object on this level?
                bne LCF72                       ; brif not - look for another object
                andcc #$fb                      ; turn off Z flag for object found
LCF80           puls a,pc                       ; restore registers and return
LCF82           ldx #creaturetab-17             ; point to creature table
LCF85           leax $11,x                      ; move to next entry
                cmpx #mazedata                  ; end of table?
                beq LCF96                       ; brif so
                cmpd 15,x                       ; is the creature in the desired maze location
                bne LCF85                       ; brif not - check another
                tst 12,x                        ; is the creature alive?
                beq LCF85                       ; brif not - check another
LCF96           rts                             ; return to caller, Z clear if we found a creature
LCF97           pshs a,b,x                      ; save registers
LCF99           jsr LCC71                       ; get a starting point for the creature
                std ,s                          ; save resulting location
                lda ,x                          ; fetch room data at location
                inca                            ; is it a room?
                beq LCF99                       ; brif not - try again
                puls a,b,x,pc                   ; restore registers, return value, and return
; Create a creature
```

```
LCFA5           pshs a,b,x,y,u                  ; save registers
LCFA7           ldu #creaturetab-17             ; point to creature table
LCFAA           leau $11,u                      ; move to next entry
                tst 12,u                        ; is creature alive?
                bne LCFAA                       ; brif not - look for another entry
                dec 12,u                        ; mark creature alive
                sta 13,u                        ; set creature type as requested
                ldb #8                          ; 8 bytes per creature data
                mul                             ; get offset into creature data table
                addd #LDABB                     ; now we have a pointer to this creatures data
                tfr d,y                         ; put creature data pointer in Y (source pointer)
                tfr u,x                         ; put creature slot into X (destination pointer)
                lda #8                          ; there are 8 bytes for each creature info
                jsr LC04B                       ; copy data into this creature slot
LCFC4           bsr LCF97                       ; get a location to start the creature in
                bsr LCF82                       ; check if there's already a creature there
                bne LCFC4                       ; brif so - try again
                std 15,u                        ; put the creature there
                tfr u,x                         ; save creature pointer
                jsr LC25C                       ; get scheduling entry
                stx 5,u                         ; save creature pointer in scheduling entry
                ldd #LD041                      ; creature scheduling handler
                std 3,u                         ; set handler for this entry
                lda 6,x                         ; get scheduling ticks for creature
                ldb #4                          ; put in 10Hz list
                jsr LC21D                       ; go add to scheduling list
                puls a,b,x,y,u,pc               ; restore registers and return
LCFE1           pshs a,b,x,u                    ; save registers
                ldu holetabptr                  ; point to hole table for this level (going up)
                bsr LCFF2                       ; see if there is a hole for this room
                tsta                            ; is there a hole?
                bpl LCFEE                       ; brif so - return info
                bsr LCFF2                       ; check for this level going down
                adda #2                         ; flag the hole as down
LCFEE           sta ,s                          ; save hole information for return
                puls a,b,x,u,pc                 ; restore registers and return
LCFF2           lda ,u+                         ; fetch hole flags
                bmi LCFFC                       ; brif end of table entries
                ldx ,u++                        ; get location for the hole
                cmpx 2,s                        ; does it match the current location?
                bne LCFF2                       ; brif not - try another entry
LCFFC           rts                             ; return to caller
; This is the "hole/ladder" table. Each entry is suffixed by $80. Each set specifies the
; holes and ladders between two levels. The first is between levels 1 and 2. The second is
; between levels 2 and 3. And so on. You will not that the table includes references to
; level 0 (above the dungeon) and level 6 (below the dungeon) - they are simply empty
; table entries which prevents having to have special cases to handle them.
holetab         fcb $80                         ; marker for end of "level 0" to level 1
                fcb 1,0,23
                fcb 0,15,4
                fcb 0,20,17
                fcb 1,28,30
                fcb $80                         ; marker for end of level 1-2
                fcb 1,2,3
                fcb 0,3,31
                fcb 0,19,20
                fcb 0,31,0
                fcb $80                         ; marker for end of level 2-3
                fcb $80                         ; marker for end of level 3-4
                fcb 0,0,31
                fcb 0,5,0
                fcb 0,22,28
                fcb 0,31,16
                fcb $80                         ; marker for end of level 4-5
                fcb $80                         ; marker for end of level 5-6
; This is the routine that adjusts the creature counts for handling retreats. It is called every
; five minutes. If there are less than 32 creatures on the current level, it will pick a random
; creature (club giants through galdrogs) and bump the count that will be spawned the next time
; the level is entered. This *only* applies to the level currently being played.
;
; It's worth noting that this can ONLY affect levels 1, 2, and 3 because there is no way to return
; to levels 4 (no holes up from 5) which means  level 5 can only be entered once.
LD027           ldx creaturecntptr              ; point to creature counts for this level
                ldb #11                         ; maximum creature number
                clra                            ; initialize count
LD02C           adda b,x                        ; add the number of this creature
                decb                            ; at end of creature list?
                bpl LD02C                       ; brif not
                cmpa #32                        ; do we have the maximum number of creatures yet?
                bhs LD03D                       ; brif so
                getrandom                       ; get a random value
                anda #7                         ; only interested in spawning one of 8 creatures
                adda #2                         ; offset above vipers
                inc a,x                         ; bump creature count for that type
LD03D           ldd #$0508                      ; reschedule for 5 minutes
                rts                             ; return to caller
; This is the routine that handles creature movement, etc.
LD041           ldy 5,u                         ; get creature data pointer
                tst creaturefreeze              ; are creatures frozen (after the Wizard is beaten)?
                bne LD06A                       ; brif so
                ldb 12,y                        ; is the creature alive?
                bne LD04D                       ; brif so
                rts                             ; return to caller
LD04D           lda 13,y                        ; get the creature type
                cmpa #6                         ; is it a scorpion?
                beq LD06D                       ; brif so
                cmpa #10                        ; is it the wizard's image or wizard?
                bge LD06D                       ; brif so
                ldd 15,y                        ; fetch room location
                clr objiterstart                ; reset object iterator
                jsr LCF53                       ; fetch first object in room
                beq LD06D                       ; brif no object in room
                ldd 8,y                         ; get creature inventory pointer
                stx 8,y                         ; save room object as head of inventory list
                std ,x                          ; save inventory list as next item
                dec 5,x                         ; mark object as carried
                updatedungeon                   ; update the dungeon view
LD06A           jmp LD103                       ; go reschedule
LD06D           ldd 15,y                        ; get cerature location
                cmpd playerloc                  ; is it in the room with the player?
                bne LD0B2                       ; brif not
                lda 13,y                        ; get creature type
                ldb #$ff                        ; maximum sound volume
                playsound                       ; go make the creature sound (always makes on attack)
                ldd #$8080                      ; base defense modifiers
                ldx lefthand                    ; get object in left hand
                bsr LD09E                       ; set modifiers if shield
                ldx righthand                   ; get object in right hand
                bsr LD09E                       ; set modifiers if shield
                sta magicdef                    ; save magical defense value for player
                stb physdef                     ; save physical defense value for player
                tfr y,x                         ; put the creature as the attacker
                ldu #powerlevel                 ; put the player as defender
                jsr attack                      ; calculate an attack
                bmi LD099                       ; brif attack failed
                playsoundimm $13                ; play the hit sound
                jsr damage                      ; go damage the player
LD099           checkdamage                     ; check damage levels
                jmp LD10F                       ; go reschedule
LD09E           pshs a,b,x                      ; save registers
                beq LD0B0                       ; brif no object
                lda 10,x                        ; get object type
                cmpa #3                         ; is it a shield?
                bne LD0B0                       ; brif not
                ldx 6,x                         ; get magical and physical defense values
                cmpx ,s                         ; is it higher (magic has precedence)
                bhs LD0B0                       ; brif so - less good
                stx ,s                          ; save new defense multipliers
```

```
LD0B0           puls a,b,x,pc                   ; restore registers and return
LD0B2           cmpa playerloc                  ; are we in the same horizontal line as the player?
                bne LD0C3                       ; brif not
                lda 16,y                        ; get vertical coordinate for creature
                ldb #1                          ; assume east
                suba playerloc+1                ; calculate distance to player
                bmi LD0D0                       ; brif negative - player is east
                ldb #3                          ; player is actually west
                bra LD0D0                       ; go check movement
LD0C3           ldd 15,y                        ; get creature location
                cmpb playerloc+1                ; are we in the same column as the player?
                bne LD0E4                       ; brif not
                ldb #2                          ; assume south
                suba playerloc                  ; calculate difference to player
                bmi LD0D0                       ; brif player is south
                clrb                            ; set north
LD0D0           stb curdir                      ; save direction
                ldd 15,y                        ; get creature location
LD0D4           bsr LD136                       ; calculate new coordinates
                bne LD0E4                       ; brif not a room
                cmpd playerloc                  ; is the new room the player's place?
                bne LD0D4                       ; brif not
                ldb curdir                      ; get direction to move
                stb 14,y                        ; set last movement direction to player direction
                clrb                            ; select a last ditch direction
                bra LD101                       ; go try the movement and continue
LD0E4           ldx #LD114                      ; point to direction selections
                getrandom                       ; fetch a random value

                tsta                            ; set flags
                bmi LD0EE                       ; brif negative
                leax 3,x                        ; select alternative direction sets
LD0EE           anda #3                         ; normalize direction to 0-3
                bne LD0F4                       ; brif nonzero
                leax 1,x                        ; move to next value
LD0F4           lda #3                          ; try 3 times for a movement
LD0F6           ldb ,x+                         ; get direction modifier
                bsr LD14F                       ; go handle movement
                beq LD103                       ; brif movement succeeded
                deca                            ; have we tried enough times?
                bne LD0F6                       ; brif not
                ldb #2                          ; try one more last ditch option
LD101           bsr LD14F                       ; do movement
LD103           lda 6,y                         ; get movement tick rate
                ldx 15,y                        ; get creature location
                cmpx playerloc                  ; does it match the player?
                bne LD111                       ; brif not - use movement rate
                updatedungeon                   ; update the dungeon display immediately
                clr dungeonchg                  ; mark dungeon update not required
LD10F           lda 7,y                         ; get attack tick rate
LD111           ldb #4                          ; and schedule for the 10Hz timer
                rts                             ; return to caller
LD114           fcb $00,$03,$01,$00,$01,$03,$00 ; direction rotations for movement choices
LD11B           pshs a,b                        ; save coordinates
                ldb curdir                      ; get direction to move
                andb #3                         ; force it to 0-3
                lslb                            ; two bytes per direction adjuster
                ldx #LD12E                      ; point to direction adjusters
                ldd b,x                         ; get adjuster
                adda ,s+                        ; apply north/south adjustment
                addb ,s+                        ; apply east/west adjustment
                jmp LCC7B                       ; convert to pointer in X
LD12E           fdb $ff00                       ; move north (-1, 0)
                fdb 1                           ; move east (0, +1)
                fdb $100                        ; move south (+1, 0)
                fdb $ff                         ; move west (0, -1)
LD136           pshs a,b,x,y,u                  ; save registers
                bsr LD11B                       ; calculate new coordinates
                jsr LCC8E                       ; check if we fell off map
                bne LD14D                       ; brif so
                tfr d,u                         ; save coordinates for later
                lda ,x                          ; get data at the new location
                inca                            ; is it a room?
                beq LD14C                       ; brif not
                stu ,s                          ; save new coordinates for return
                stx 2,s                         ; save new room pointer
                lda #1                          ; set so we get Z=1 on return
LD14C           deca                            ; set flags for success/fail
LD14D           puls a,b,x,y,u,pc               ; restore registers and return
LD14F           pshs a,b,x                      ; save registers
                addb 14,y                       ; add selected rotation to current movement direction
                andb #3                         ; normalize to 0-3
                stb curdir                      ; save new direction
                ldd 15,y                        ; get creature location
                bsr LD136                       ; calculate new coordinates
                bne LD199                       ; brif no room there
                jsr LCF82                       ; get creature in room
                bne LD199                       ; brif there's a creature there - can't go
                std 15,y                        ; save new creature location
                ldb curdir                      ; get direction
                stb 14,y                        ; save as last moved direction
                ldd 15,y                        ; get new location
                suba playerloc                  ; get distance from player (Y)
                bpl LD16F                       ; brif positive
                nega                            ; invert msb (absolute value)
LD16F           subb playerloc+1                ; get distance from player (X)
                bpl LD174                       ; brif positive
                negb                            ; invert lsb (absolute value)
LD174           stb accum0                      ; save X distance
                cmpa accum0                     ; is the Y distance more?
                bge LD17C                       ; brif so
                exg a,b                         ; use the Y distance then
LD17C           sta accum0                      ; save calculated distance
                cmpa #8                         ; more than 8 rooms away in long distance?
                bgt LD198                       ; brif so
                cmpb #2                         ; more than 2 rooms away in short distance?
                bgt LD198                       ; brif so
                getrandom                       ; get a random value
                bita #1                         ; do we need to make a sound?
                beq LD196                       ; brif we won't make a sound
                lda accum0                      ; get distance
                ldb #$1f                        ; multplier for distance
                mul                             ; calculate distance volume modifier
                comb                            ; invert it so closer is louder
                lda 13,y                        ; get creature number
                playsound                       ; go make the creature's sound
LD196           dec dungeonchg                  ; mark dungeon update required
LD198           clra                            ; set Z for movement happened
LD199           puls a,b,x,pc                   ; restore registers and return
; This is the routine that ticks down the torch.
LD19B           ldu curtorch                    ; get currently burning torch
                beq LD1BC                       ; brif no torch in use
                lda 6,u                         ; get remaining torch life
                beq LD1BC                       ; brif already empty
                deca                            ; reduce time remaining
                sta 6,u                         ; update object data
                cmpa #5                         ; is it 5 minutes left?
                bgt LD1B0                       ; brif more
                ldb #$18                        ; object type "DEAD TORCH"
                stb 9,u                         ; set torch to DEAD TORCH
                clr 11,u                        ; mark as fully revealed
LD1B0           cmpa 7,u                        ; is time remaining less than physical light strength?
                bge LD1B6                       ; brif not
                sta 7,u                         ; tick down physical light strength
LD1B6           cmpa 8,u                        ; is time remaining less than magical light strength?
                bge LD1BC                       ; brif not
                sta 8,u                         ; tick down magical light strength
LD1BC           dec dungeonchg                  ; mark update to dungeon required
                ldd #$0108                      ; reschedule for one minute
                rts                             ; return to caller
; This is the routine that periodically updates the dungeon display (or scroll). It does not update
; unless something has marked the display changed OR a scroll is being displayed. It is called twice
; per second.
LD1C2           tst dungeonchg                  ; check if we need to update dungeon display
                bne LD1CD                       ; brif so
                ldx #displayscroll              ; are we displaying a scroll?
                cmpx displayptr
                bne LD1D1                       ; brif not
LD1CD           clr dungeonchg                  ; mark update not required
                updatedungeon                   ; update dungeon display
LD1D1           ldd #$0304                      ; reschedule check for 0.5 seconds
                rts                             ; return to caller

LD1D5           clra                            ; set NULL value
                clrb
                subd damagelevel                ; subtract it from the current damage level
                jsr asrd6                       ; shift right 6 bits (divide by 64)
                addd damagelevel                ; reduce damage level by 1/64 of original damage level
                bgt LD1E2                       ; brif new damage level > 0
                clra                            ; minimize damage level at 0
                clrb
LD1E2           std damagelevel                 ; save new damage level
                checkdamage                     ; check damage level and calculate ticks until next recovery
run
                lda heartticks                  ; get ticks to reduce damage (heart rate)
                ldb #2                          ; requeue in the 60Hz ticker
                rts                             ; return to caller

; This routine handles the keyboard input.
LD1EB           tst waitnewgame                 ; are we waiting for a new game?
                bne LD21B                       ; brif so
LD1EF           jsr readkeybuf                  ; get a key from buffer
                tsta                            ; did we get something?
                beq LD248                       ; brif not
                tst nokeyboard                  ; is keyboard disabled?
                bne LD1EF                       ; brif so - keep draining buffer
                cmpa #$20                       ; is it a space?
                beq LD215                       ; brif so
                ldb #$1f                        ; value for CR
                cmpa #$0d                       ; is it CR?
                beq LD212                       ; brif so
                ldb #$24                        ; value for BS
                cmpa #8                         ; is it BS?
                beq LD212                       ; brif so
                clrb                            ; value for nothing (space)
                cmpa #$41                       ; is it a letter?
                blo LD212                   ; brif below uppercase alpha
                cmpa #$5a                   ; is it still a letter?
                bls LD215                   ; brif uppercase alpha
LD212           tfr b,a                     ; save calculated code
                skip2                       ; skip instruction
LD215           anda #$1f                   ; normalize down to 0...31
                bsr LD24C
                bra LD1EF                   ; go handle another character
LD21B           ldy demoseqptr              ; fetch pointer to command sequence
                ldb ,y+                     ; do we have a command to do?
                bpl LD229                   ; brif so
                delay                       ; wait for a bit
                delay                       ; wait for a bit more
                jmp START                   ; go start over again with the splash and demo
LD229           ldx ,y++                    ; get pointer to the word
                ldu #cmddecodebuff          ; point to command decode buffer
                decodestr                   ; decode the keyword
                leau 1,u                    ; move past the "object type" flag
                delay                       ; wait a bit
                skip2                       ; skip next instruction
LD235           bsr LD24C                   ; go handle input character
                lda ,u+                     ; fetch a character from the decoded string
                bpl LD235                   ; brif not end of string
                clra                        ; code for a space
                bsr LD24C                   ; go handle input character
                decb                        ; have we consumed all the words in this command?
                bne LD229                   ; brif not - get another
                lda #$1f                    ; code for carriage return
                bsr LD24C                   ; add character to buffer and process if needed
                sty demoseqptr              ; save new command stream pointer
LD248           ldd #$0102                  ; reschedule for next tick
                rts                         ; return to caller
LD24C           pshs a,b,x,y,u              ; save registers
                tst hidestatus              ; are we starting a new command string?
                bne LD256                   ; brif not
                resetdisplay                ; clear command area, reset status, and redisplay dungeon
                showprompt                  ; show the prompt
LD256           ldu linebuffptr             ; get input buffer pointer
                cmpa #$1f                   ; end of line?
                beq LD26F                   ; brif so
                cmpa #$24                   ; BS?
                beq LD27D                   ; brif so
                renderchar                  ; render the character
                sta ,u+                     ; save in buffer
                ldx #LC67C                  ; point to cursor string
                renderstr                   ; go render the cursor
                cmpu #linebuffend               ; is the buffer full?
                bne LD2B4                       ; brif not
LD26F           clra                            ; make a space
                renderchar                      ; render it
                ldd allones                     ; get end of string marker
                std ,u++                        ; save in buffer
                ldu #linebuff                   ; reset buffer pointer to start of line
                stu linebuffptr                 ; save new buffer pointer
                bra LD292                       ; go process command
LD27D           cmpu #linebuff                  ; are we at the start of the line?
                beq LD2B4                       ; brif so - BS does nothing
                leau -1,u                       ; move buffer pointer back
                ldx #LD28C                      ; pointer to SPACE BS BS _ BS
                renderstr                       ; display the backspace string
                bra LD2B4                       ; get on with things
LD28C           fcb $00,$24,$24,$1c,$24,$ff     ; unpacked SPACE BS BS _ BS string
LD292           ldx #kwlist_cmd                 ; point to command list
                jsr LCBEC                       ; look up word in command list
                beq LD2A7                       ; brif nothing to match
                bpl LD2A1                       ; brif found
                jsr badcommand                  ; show bad command string
                bra LD2A7                       ; go on with new command
LD2A1           lsla                            ; two bytes per jump table entry
                ldx #LD9D0                      ; point to command jump table
                jsr [a,x]                       ; go handle command
LD2A7           ldu #linebuff                   ; start of command buffer
                tst hidestatus                  ; have we been told to start a new command stream?
                beq LD2B4                       ; brif so - don't display prompt

                tst nokeyboard                  ; is keyboard disabled?
                bne LD2B4                       ; brif so - no prompt
                showprompt                      ; show a new prompt
LD2B4           stu linebuffptr                 ; save new buffer pointer
                puls a,b,x,y,u,pc               ; restore registers and return
cmd_attack      jsr LCC31                       ; get pointer to specified hand
                ldu ,u                          ; fetch item in specified hand
                bne LD2C2                       ; brif item there
                ldu #emptyhand                  ; point to data for emtpy hand
LD2C2           tfr u,y                         ; save object data pointer
                lda 12,u                        ; fetch magical offense value
                sta magicoff                    ; save for combat calculations
                lda 13,u                        ; fetch physical offense value
                sta physoff                     ; save for combat calculations
                adda magicoff                   ; calculate sum of magical and physical damage
                rora                            ;* divide by 8
                lsra                            ;*
                lsra                            ;*
                ldx powerlevel                  ; fetch current player power
                jsr applyscale                  ; apply the scale factor calculated above
                addd damagelevel                ; apply the wielding cost to play damage
                std damagelevel                 ; save new damage value
                lda 10,u                        ; get object type
                adda #12                        ; offset into sound table
                ldb #$ff                        ; set full volume
                playsound                       ; play the attack sound for the object
                lda 9,u                         ; get object subtype
                cmpa #$13                       ; is it less than "ENERGY"?
                blt LD2F7                       ; brif so - not an expiring ring
                cmpa #$15                       ; is it more than "FIRE"?
                bgt LD2F7                       ; brif so - not an expiring ring
                dec 6,u                         ; count down ring usages
                bne LD2F7                       ; brif not used up
                lda #$16                        ; type for "GOLD"
                sta 9,u                         ; set to GOLD ring
                jsr LD638                       ; update object stats appropriately
LD2F7           ldd playerloc                   ; get current location in dungeon
                jsr LCF82                       ; find creature in the room
                beq LD375                       ; brif no creature
                ldu #powerlevel                 ; point to player power level
                exg x,u                         ; swap player and creature pointers
                lda 10,y                        ; fetch object type
                cmpa #1                         ; is it a ring?
                beq LD31F                       ; go do successful attack if so - rings never miss
                jsr attack                      ; calculate if attack succeeds (attacker in X, defender in U)
                bmi LD375                       ; brif attack fails
                ldy curtorch                    ; do we have a torch burning?
                beq LD319                       ; brif not
                lda 9,y                         ; get torch type
                cmpa #$18                       ; is it "DEAD"?
                bne LD31F                       ; brif not
LD319           getrandom                       ; get random number
                anda #3                         ; 1 in 4 chance of a hit in the dark
                bne LD375                       ; brif we didn't hit
LD31F           playsoundimm $12                ; play the "HIT" sound
                renderstrimmp                   ; display the "!!!" for a successful hit
                fcb $16,$f7,$b0                 ; packed "!!!" string
                jsr damage                      ; calculate damage, apply to victim
                bhi LD375                       ; brif not dead
                leax 8,u                        ; point to inventory head pointer
LD32E           ldx ,x                          ; get next inventory item
                beq LD33A                       ; brif end of inventory
                clr 5,x                         ; mark item as on the floor
                ldd 15,u                        ; get location of creature
                std 2,x                         ; put the object there
                bra LD32E                       ; go process next inventory item
LD33A           ldx creaturecntptr              ; point to creature count table for this level
                ldb 13,u                        ; get type of creature killed
                dec b,x                         ; reduce number of this creature type
                clr 12,u                        ; flag creature as dead
                updatedungeon                   ; update the dungeon display
                playsoundimm $15                ; play the "kill" sound
                ldd ,u                          ; fetch creature power level
                bsr asrd3                       ; divide by 8
                addd powerlevel                 ; add gained power to current power level
                bpl LD351                       ; brif power level did not overflow
                lda #$7f                        ; maximize power level at 32767
LD351           std powerlevel                  ; save adjusted power level for player
                lda 13,u                        ; get the dead creature type
                cmpa #10                        ; is dead creature wizard's image?
                beq LD386                       ; brif so - do the annoyed wizard
                cmpa #11                        ; is dead creature the wizard?
                bne LD375                       ; brif not
                dec creaturefreeze              ; stop the creatures
                ldd #$713                       ; constants for physical light 7, magical light 19
                std baselight                   ; set base light level in dungeon
                ldx #objecttab+14               ; pointer to second object slot in object table
                stx objectfree                  ; mark end of object table at just past first object
                ldd zero                        ; NULL pointer
                std backpack                    ; mark backpack empty
                std curtorch                    ; mark no torch burning
                std righthand                   ; mark right hand empty
                std lefthand                    ; mark left hand empty
                resetdisplay                    ; reset display and show dungeon
LD375           checkdamage                     ; update the damage situation
; The following are pointless in this routine - we're returning from a command and D is zero anyway!
asrd7           asra                            ; enter here to do an arithmetic right shift 7 bits
                rorb
asrd6           asra                            ; enter here to do an arithmetic right shift 6 bits
                rorb
asrd5           asra                            ; enter here to do an arithmetic right shift 5 bits
                rorb
asrd4           asra                            ; enter here to do an arithmetic right shift 4 bits
                rorb
asrd3           asra                            ; enter here to do an arithmetic right shift 3 bits
                rorb
                asra
                rorb
                asra
                rorb
                rts                             ; return to caller
LD386           ldx #img_wizard                 ; point to Wizard graphic
                fadeinclrst                     ; fade in the wizard
                renderstrimmp                   ; dipslay "ENOUGH! I TIRE OF THIS PLAY..."
                fcb $ff,$c0,$57,$3e             ; packed string "ENOUGH! I TIRE OF THIS PLAY..."
                fcb $a7,$46,$c0,$90
                fcb $51,$32,$28,$1e
                fcb $60,$51,$09,$98
                fcb $20,$c0,$e7,$de
                fcb $f0
                renderstrimmp                   ; also display "PREPARE TO MEET THY DOOM!!!"
                fcb $e8,$00,$08,$48             ; packed string "PREPARE TO MEET THY DOOM!!!"
                fcb $b0,$0c,$8a,$0a
                fcb $3c,$0d,$29,$68
                fcb $0a,$23,$20,$23
                fcb $de,$dd,$ef,$60
                delay                           ; delay a bit
                ldu curtorch                    ; fetch current torch
                stu backpack                    ; put it in the backpack
                beq LD3C4                       ; brif no torch
                clra                            ; make sure the torch is the only thing in the backpack
                clrb
                std ,u
LD3C4           ldd #200                        ; set player carry weight to 200
                std carryweight
                lda #3
                createlevel
                jsr LCF97
                std playerloc
                fadeout                         ; fade out the wizard
                resetdisplay
                rts
; Calculate the probability of a successful hit.
; Enter with the attacker info pointed to be X and the defender data pointed to by U.
;
; It first does the following calculation:
; MAX(15-(4(DPOW-DDAM)/APOW),0)
; 4(DPOW-DDAM)/APOW yields a fraction which is < 4 if the defender's remaining health is
; less than the attacker's power or > 4 if the defender's remaining health is greater
; than the attacker's power. This ranges from 0% to 375% in steps of 25%.
; This result is subtracted from 15 so that low numbers mean the attacker relatively weaker
; and higher numbers mean the attacker is relatively stronger. The final range is from 0
; (where the defender is much stronger than the attacker) to 15 where the attacker is very
; much stronger than the defender.
;
; These values are converted to a signed 16 bit number. Then an 8 bit unsigned random number
; is added to the result. Finally, 127 is subtracted. If the final result is < 0, then the
; attack fails. Otherwise, the attack succeeds.
;
; The following chart gives calculation results. V is the result of MAX(...) calculation
; above. Pb is the base value calculated by the routine. Rl is the low end of the range
; of the result once the random number is applied and the 127 is subtracted. Rh is the
; high end of the range. Finally, P% is the chance of a successful hit for that result.
;
; V     Pb      Rl      Rh      P%
; 0     -75     -202    53      21.1
; 1     -50     -177    78      30.9
; 2     -25     -152    103     40.6
; 3     0       -127    128     50.4
; 4     10      -117    138     54.3
; 5     20      -107    148     58.2
; 6     30      -97     158     62.1
; 7     40      -87     168     66.0
; 8     50      -77     178     69.9
; 9     60      -67     188     73.8
; 10    70      -57     198     77.7
; 11    80      -47     208     81.6
; 12    90      -37     218     85.5
; 13    100     -27     228     89.5
; 14    110     -17     238     93.4
; 15    120     -7      248     97.3
;
; As you can see, the lower 4 values are on a steeper slope than the remaining values.
; Otherwise, the scale is perfectly linear. Also, the worst chance of success, no matter
; how overmached, is 21.1%. The best chance, no matter how much stronger the attacker,
; is less than 100%.
attack          pshs a,b,x,u                    ; save registers
                lda #15                         ; maximum value of the V calculation
                sta accum0                      ; initialze V accumulator
                ldd ,u                          ; get victim power level
                subd 10,u                       ; get difference between that and victim damage level (health)
                jsr LCA12                       ; multiply difference by 4
LD3E4           subd ,x                         ; subtract attackers power
                bcs LD3EC                       ; brif we wrapped - we have our quotient
                dec accum0                      ; count down quotient
                bne LD3E4                       ; brif we haven't counted down to nothing
LD3EC           ldb accum0                      ; get result (V as above)
                subb #3                         ; one of first three values?
                bpl LD3FB                       ; brif not
                negb                            ; now 0 became 3, 1 became 2, and 2 became 1
                lda #$19                        ;* multiply by factor (25)
                mul                             ;*
                jsr LCA99                       ; negate result (-75, -50, and -25)
                bra LD3FE                       ; calculate attack
LD3FB           lda #10                         ;* multiply by factor (10) (all others are linear going up by
10
                mul                             ;* for each step
LD3FE           std ,--s                        ; save probability base
                getrandom                       ; get a random value
                tfr a,b                         ; save random value
                clra                            ; zero extend
                addd ,s++                       ; add to probabilty base
                subd #$7f                       ; subtract 127 so that >= 0 is a hit, < 0 is a miss
                puls a,b,x,u,pc                 ; restore registers and return
; This routine calculates the damage done by an attack. Enter with the attacker info at X and the defender
; info at U.
damage          pshs a,b,x,y,u                  ; save registers
                tfr x,y                         ; save attacker pointer
                ldx ,y                          ; get attacker power
                lda 2,y                         ; get magical offsense power
                bsr applyscale                  ; scale it
                tfr d,x                         ; save result
                lda 3,u                         ; get defender magical defense
                bsr applyscale                  ; scale it
                addd 10,u                       ; add in defenders current damage
                std 10,u                        ; save new defender damage
                ldx ,y                          ; get attacker power
                lda 4,y                         ; get physical offense power
                bsr applyscale                  ; scale it
                tfr d,x                         ; save it

                lda 5,u                         ; get defender's physical defense power
                bsr applyscale                  ; scale it
                addd 10,u                       ; add to current defender damage level
                std 10,u                        ; save new damage level
                ldx ,u                          ; get defender's power
                cmpx 10,u                       ; compare with new damage level
                puls a,b,x,y,u,pc               ; restore registers and return
; Multiply X by the value in A, where the binary point in A is to the left of bit 6. Return only the
; integer result in D (rounded down).
applyscale      pshs a,b,x                      ; save parameters and registers
                clr accum0                      ; blank out temp storage area
                ldb 3,s                         ; get LSB of X
                mul                             ; multiply LSB
                std accum0+1                    ; save in scratch variable
                lda ,s                          ; fetch muliplier
                ldb 2,s                         ; fetch MSB of X
                mul                             ; multiply it
                addd accum0                     ; add in partial product
                lsl accum0+2                    ;* shift product left so binary point is to the right of
                rolb                            ;* of the upper 16 bits - leave interger result in D.
                rola                            ;*
                std ,s                          ; save integer result for return
                puls a,b,x,pc                   ; clean up parameters, fetch product, and return
cmd_climb       ldd playerloc                   ; get player location
                jsr LCFE1                       ; fetch hole information
                bmi LD46F                       ; brif no holes
                sta accum0                      ; save hole info
                ldx #kwlist_dir                 ; point to direction list
                jsr LCBEC                       ; go parse direction
                ble LD46F                       ; brif no direction
                ldb accum0                      ; get hole info
                cmpa #4                         ; is it up?
                beq LD472                       ; brif so
                cmpa #5                         ; is it down?
                bne LD46F                       ; brif not
                lda #1                          ; level goes up one if we descend
                bitb #2                         ; is there a hole down?
                bne LD478                       ; brif so
LD46F           jmp badcommand                  ; complain about bad direction or no hole
LD472           lda #$ff                        ; level goes down one if we ascend
                cmpb #1                         ; do we have a ladder?
                bne LD46F                       ; brif not
LD478           showprepare                     ; show the scary PREPARE! screen
                adda currentlevel               ; calculate the new level number
                createlevel                     ; build the new level
                resetdisplay                    ; reset everything and show the maze
                rts                             ; return to caller
cmd_examine     ldx #LD495                      ; pointer to the inventory display routine
                stx displayptr                  ; set up the display update routine
                updatedungeon                   ; update the display
                rts                             ; return to caller
LD489           cleargfx2                       ; clear graphics
                ldx ,u                          ; get current text area start
                ldu #infoarea                   ; point to info text area descriptor
                stx ,u                          ; set text area start to the same place
                dec textother                   ; set to nonstandard text rendering
                rts                             ; return to caller
; This is the dungeon display routine that handles showing the inventory list.
LD495           bsr LD489                       ; clear the graphics area and set up for text rendering
                clr columnctr                   ; flag column zero in object list
                ldd #10                         ;* set up to centre "IN THIS ROOM"
                std 4,u                         ;* column 10, row 0
                renderstrimmp                   ; show the "IN THIS ROOM" heading
                fcb $62,$5c,$0a,$21             ; packed string "IN THIS ROOM"
                fcb $33,$04,$9e,$f6
                fcb $fc
                ldd playerloc                   ; get player location
                jsr LCF82                       ; get creature at player location
                beq LD4C0                       ; brif no creature there
                ldx 4,u                         ; get current text position
                leax 11,x                       ; move 11 over
                stx 4,u                         ; save new position
                renderstrimmp                   ; show the "!CREATURE!" string if a creature is present
                fcb $56,$c7,$22,$86             ; packed string "!CREATURE!"
                fcb $95,$91,$77,$f0
LD4C0           clr objiterstart                ; reset object iterator
LD4C2           ldd playerloc                   ; get player location
                jsr LCF53                       ; fetch next object
                beq LD4CD                       ; brif no more objects
                bsr LD505                       ; display object
                bra LD4C2                       ; go handle another object
LD4CD           tst columnctr                   ; are we at the start of a line?
                beq LD4D3                       ; brif so
                bsr LD4FE                       ; do a newline
LD4D3           ldd #$1b20                      ; set up for displaying a row of !!!!
LD4D6           renderchar                      ; display a !
                decb                            ; done enough of them?
                bne LD4D6                       ; brif not
                ldx 4,u                         ; get current text location
                leax 12,x                       ; adjust for centering
                stx 4,u                         ; save new text location
                renderstrimmp                   ; display "BACKPACK" heading

                fcb $40,$82,$35,$c0             ; packed string "BACKPACK"
                fcb $23,$5f,$c0
                ldx #backpack                   ; point to backpack head pointer
LD4ED           ldx ,x                          ; get next item in backpack
                beq LD4FB                       ; brif nothing else in backpack
                cmpx curtorch                   ; is the object the currently burning torch?
                bne LD4F7                       ; brif not
                com 6,u                         ; invert video if it is
LD4F7           bsr LD505                       ; display ojbect name
                bra LD4ED                       ; go display another object
LD4FB           clr textother                   ; reset to standard text rendering
                rts                             ; return to caller
LD4FE           lda #$1f                        ; character code for newline
                renderchar                      ; go move to next line
                clr columnctr                   ; flag column 1
                rts                             ; return to caller
LD505           pshs a,b,x                      ; save registers
                jsr LC617                       ; fetch object name string (decoded)
                renderstr                       ; display object name
                lda levbgmask                   ; get current level mask
                sta 6,u                         ; restore proper background
                com columnctr                   ; are we on column 1 or 2?
                beq LD51E                       ; brif back at column 1
                ldd 4,u                         ; get cursor position
                addd #$10                       ; move right 16 cells
                andb #$f0                       ; round down to multiple of 16
                std 4,u                         ; save new cursor position
                skip2                           ; move on with routine
LD51E           bsr LD4FE                       ; do a newline
                puls a,b,x,pc                   ; restore registers and return
cmd_get         bsr LD576                       ; go parse hand and return pointer to it
                bne LD573                       ; brif no direction
                jsr parseobj                    ; go parse an object
                clr objiterstart                ; reset object iterator
LD52B           ldd playerloc                   ; get current dungeon location
                jsr LCF53                       ; fetch next object
                beq LD573                       ; brif no more objects
                tst parsegenobj                 ; did we get a generic object type?
                bne LD53C                       ; brif not
                lda 10,x                        ; get object type we're looking at
                cmpa parseobjtypegen            ; does it match?
                bra LD540                       ; go finish up
LD53C           lda 9,x                         ; get specific object type
                cmpa parseobjtype               ; does it match?
LD540           bne LD52B                       ; brif not - try another
                stx ,u                          ; put object in selected hand

                inc 5,x                         ; mark as not on floor
                ldb 10,x                        ; get object general type
                ldx #LD9FA                      ; point to weight table
                ldb b,x                         ; get object weight
                clra                            ; zero extend
                bra LD56B                       ; go adjust carried weight
cmd_drop        bsr LD576                       ; parse a hand and get pointer
                beq LD573                       ; brif no hand
                clra                            ; NULL Pointer
                clrb
                std ,u                          ; empty the hand out
                clr 5,x                         ; mark object as on floor
                ldd playerloc                   ; get dungeon location
                std 2,x                         ; set object location
                lda currentlevel                ; get current level
                sta 4,x                         ; set object level
                ldb 10,x                        ; get object general type
                ldx #LD9FA                      ; point to weight table
                ldb b,x                         ; get weight of object
                negb                            ; negate it for subtraction
                sex                             ; sign extend
LD56B           addd carryweight                ; add weight adjustment to carried weight
                std carryweight                 ; save new carried weight
                checkdamage                     ; go update the damage situation
                bra LD5B7                       ; update display and return
LD573           jmp badcommand                  ; complain about bad command
LD576           jmp LCC31                       ; go parse a hand and return pointer
cmd_stow        bsr LD576                       ; get pointer to object in requested hand
                beq LD573                       ; brif no object in the hand
LD57D           ldd backpack                    ; get first item in backpack
                std ,x                          ; make it the next item in the list
                stx backpack                    ; make this item the first item in the backpack
                clra                            ; NULL pointer
                clrb
                std ,u                          ; mark selected hand empty
                bra LD5B7                       ; update status line, etc.
cmd_pull        bsr LD576                       ; fetch pointer to object in specified hand
                bne LD573                       ; brif there is something in that hand
                jsr parseobj                    ; parse object name
                ldx #backpack                   ; point to backpack head pointer
LD593           tfr x,y                         ; save previous pointer location
                ldx ,x                          ; fetch pointer to next item
                beq LD573                       ; brif end of list
                tst parsegenobj                 ; is a specific object type requested?
                bne LD5A3                       ; brif so
                lda 10,x                        ; get object type (general) requested
                cmpa parseobjtypegen            ; does the object match?
                bra LD5A7                       ; finish up the loop
LD5A3           lda 9,x                         ; get object type (specific) requested
                cmpa parseobjtype               ; does it match requested object type?
LD5A7           bne LD593                       ; brif not matching object
                ldd ,x                          ; get next pointer
                std ,y                          ; put in previous next pointer (remove from backpack)
                stx ,u                          ; save object in the specified hand
LD5AF           clra                            ; set up NULL pointer
                clrb
                cmpx curtorch                   ; is this object the current torch?
                bne LD5B7                       ; brif not
                std curtorch                    ; turn off current torch
LD5B7           updatestatus                    ; update status line to reflect new hand contents
                updatedungeon                   ; update the dungeon display
                rts                             ; return to caller
cmd_incant      ldx #kwlist_adj                 ; point to object types list
                jsr LCBEC                       ; look up object
                ble LD5EF                       ; brif not found in list or no type specified
                tst kwexact                     ; was it a complete match?
                beq LD5EF                       ; brif not
                std parseobjtype                ; save object type
                ldu lefthand                    ; get left hand object
                bsr LD5D0                       ; check if matching object is there
                ldu righthand                   ; get right hand object and continue
LD5D0           beq LD5EF                       ; brif no object carried
                lda 10,u                        ; get general type
                cmpa #1                         ; is it a ring?
                bne LD5EF                       ; brif not
                lda 7,u                         ; get incant to type
                beq LD5EF                       ; brif there isn't one
                cmpa parseobjtype               ; does it match the one we incanted?
                bne LD5EF                       ; brif not
                sta 9,u                         ; set new type to the incanted type
                setobjectspecs                  ; reset object specs
                playsoundimm $0D                ; play the ring sound
                updatestatus                    ; update the status area
                clr 7,u                         ; mark ring as incanted
                cmpa #$12                       ; is it the FINAL ring?
                beq LD5F0                       ; brif so
LD5EF           rts                             ; return to caller
LD5F0           ldx #img_goodwiz                ; point to good wizard image
                dec enablefadesound             ; enable fade sound effect
                fadeinclrst                     ; fade in the wizard
                renderstrimmp                   ; display victory message line 1
                fcb $ff,$c4,$54,$3d             ; packed string victory message line 1
                fcb $84,$d8,$08,$59
                fcb $D1,$2e,$c8,$03
                fcb $70,$a6,$93,$05
                fcb $10,$50,$20,$2e
                fcb $20
                renderstrimmp                   ; dispaly victory message line 2
                fcb $c8,$00,$00,$00             ; packed string victory message line 2
                fcb $00,$03,$cc,$00
                fcb $81,$c5,$b8,$2e
                fcb $9d,$06,$44,$f7
                fcb $bc
LD621           bra LD621                       ; Do nothing until IRQ decides something should happen
cmd_reveal      jsr LCC31                       ; parse a hand and get pointer to hand
                ldu ,u                          ; is there an object there?
                beq LD63E                       ; brif not
                lda 11,u                        ; has object been revealed?
                beq LD63E                       ; brif so
                ldb #$19                        ; add multiplier to get needed power to reveal it
                mul                     ; multiply out
                cmpd powerlevel         ; is player strong enough?
                bgt LD63E               ; brif not
                lda 9,u                 ; fetch specific object type
LD638           setobjectspecs          ; update specs to revealed type
                clr 11,u                ; mark object as revealed
                updatestatus            ; update the status area
LD63E           rts                     ; return to caller
cmd_turn        ldx #kwlist_dir         ; point to direction list
                jsr LCBEC               ; look up word in list
                ble LD693               ; brif no match or no word
                ldb facing              ; get current direction
                cmpa #0                 ; TURN LEFT?
                bne LD654               ; brif not
                decb                    ; rotate counter clockwise
                bsr LD66D               ; normalize direction and update display
                bsr LD674               ; sweep right
                bra LD669               ; finish up
LD654           cmpa #1                 ; TURN RIGHT?
                bne LD65D               ; brif not
                incb                    ; rotate clockwise
                bsr LD66D               ; normalize direction and update display
                bra LD667               ; sweep left and finish up
LD65D           cmpa #3                 ; TURN AROUND?
                bne LD693               ; brif not
                addb #2                 ; turn 180
                bsr LD66D               ; normalize direction and update display
                bsr LD684               ; sweep left and fall through

LD667           bsr LD684               ; sweep left
LD669           dec pageswap            ; set graphic swap required
                sync                    ; wait for swap to happen
                rts                     ; return to caller
LD66D           andb #3                 ; normalize direction to 0-3
                stb facing              ; save new direction faced
                jmp LC660               ; go update display and return
LD674           bsr LD696               ; draw outline and set up for a vertical line
                bne LD683               ; brif not displaying anything
                ldd #8                  ; start at column 8
LD67B           bsr LD6BA               ; draw and erase vertical line
                addd #$20               ; move right 32 pixels
                tsta                    ; did we wrap?
                beq LD67B               ; brif not - keep going
LD683           rts                     ; return to caller
LD684           bsr LD696               ; set up for drawing the sweep
                bne LD692               ; brif we aren't drawing anything
                ldd #$f8                ; start at X coord 248
```

```
LD68B           bsr LD6BA                       ; draw and undraw the line
                subd #$20                       ; move left 32 pixels
                bpl LD68B                       ; brif we haven't wrapped yet - do another
LD692           rts                             ; return to caller
LD693           jmp badcommand                  ; carp about a bad command
LD696           ldu displayptr                  ; get display pointer
                cmpu #LCE66                     ; is it the regular dungeon display
                bne LD6B9                       ; brif not - don't show turning
                ldx #$8080                      ; scale factors of 1.0
                stx horizscale                  ; set horizontal and vertical scale factors to 1.0
                clr renderdist                  ; set render distance to 0 (immediate)
                setlighting                     ; set light level for rendering
                cleargfx1                       ; clear screen
                ldx #LD6C6                      ; point to outline graphic
                drawgraphic                     ; draw it
                ldx #$11                        ;* set start Y coord to 17
                stx ybeg                        ;*
                ldx #$87                        ;= set end Y coord to 135
                stx yend                        ;=
                clra                            ; clear Z
LD6B9           rts                             ; return to caller
LD6BA           std xbeg                        ; set start X coord
                std xend                        ; set end X coord
                bsr LD6C0                       ; draw the line and invert mask
LD6C0           jsr drawline                    ; draw the line again
                com levbgmask                   ; invert mask
                rts                             ; return to caller
; This is top and bottom lines during a turn sweep
LD6C6           fcb 16,0
                fcb 16,255
                fcb $ff
                fcb 136,0
                fcb 136,255
                fcb $fe
cmd_move        ldx #kwlist_dir                 ; point to direction list
                jsr LCBEC                       ; look up direction
                blt LD693                       ; brif bad direction
                bgt LD6E3                       ; brif there is a direction
                dec movehalf                    ; mark half step
                updatedungeon                   ; update display
                clrb                            ; set direction to forward
                clr movehalf                    ; set to normal display
                bra LD6EF                       ; go finish up
LD6E3           cmpa #2                         ; is it MOVE BACK?
                bne LD6F3                       ; brif not
                dec movebackhalf                ; set half step back
                updatedungeon                   ; go update display
                ldb #2                          ; set direction to backward
                clr movebackhalf                ; set normal display
LD6EF           bsr LD720                       ; update position
                bra LD70E                       ; go calculate movement cost, etc.
LD6F3           cmpa #1                         ; is it MOVE RIGHT?
                bne LD701                       ; brif not
                ldb #1                          ; set direction to right
                bsr LD720                       ; update position
                bne LD70E                       ; brif movement failed
                bsr LD684                       ; do a sweep left
                bra LD70E                       ; calculate movement cost, etc.
LD701           cmpa #0                         ; is it LEFT?
                bne LD693                       ; brif not
                ldb #3                          ; set direction to left
                bsr LD720                       ; update position
                bne LD70E                       ; brif movement failed
                jsr LD674                       ; do a sweep right
LD70E           ldd carryweight                 ; get current carry weight
                jsr asrd3                       ; divide by 8
                addd #3                         ; add 3 for player weight
                addd damagelevel                ; add to damage level
                std damagelevel                 ; save new damage level
                checkdamage                     ; check for pasing out
                dec pageswap                    ; set graphics swap required
                sync                            ; wait for swap to happen
                rts                             ; return to caller
LD720           pshs a,b                        ; save registers
                clr ,-s                         ; make a temp
                addb facing                     ; add direction to current facing direction
                andb #3                         ; normalize to 0-3
                stb curdir                      ; save move direction
                ldd playerloc                   ; get current player location
                jsr LD136                       ; calculate movement
                beq LD738                       ; brif movement succeeds
                playsoundimm $14                ; play the "hit the wall" sound
                dec ,s                          ; flag failed movement
                ldd playerloc                   ; get current location as result
LD738           std playerloc                   ; save new location
                jsr LC660                       ; go update the display
                tst ,s+                         ; set flags for did movement succeed?
                puls a,b,pc                     ; restore registers and return
cmd_use         jsr LCC31                       ; fetch pointer to object in specified hand
                beq LD767                       ; brif nothing in the hand
                ldd 9,x                         ; fetch object type and subtype
                cmpb #5                         ; is it a torch?
                bne LD757                       ; brif not
                stx curtorch                    ; set object as currently mounted
                jsr LD57D                       ; go place the object in the backpack
                playsoundimm $11                ; play the torch sound
                updatedungeon                   ; update dungeon with new lighting
                rts                             ; return to caller
LD757           tfr x,u                         ; save object pointer
                ldx #LD76B                      ; point to jump table
LD75C           cmpa ,x                         ; does the sub type match?
                beq LD768                       ; brif so
                leax 3,x                        ; move to next entry
                cmpx #LD77A                     ; end of table?
                blo LD75C                       ; brif not - try another
LD767           rts                             ; no match - do nothing
LD768           jmp [1,x]                       ; transfer control to specified routine
LD76B           fcb $05                         ; "THEWS" (thews flask)
                fdb LD77A
                fcb $09                         ; "HALE" (hale flask)
                fdb LD783
                fcb $08                         ; "ABYE" (abye flask)
                fdb LD787
                fcb $04                         ; "SEER" (seer scroll)
                fdb LD7A2
                fcb $07                         ; "VISION" (vision scroll)
                fdb LD7A0
LD77A           ldd #1000                       ; thews increases player power by 1000
                addd powerlevel                 ; add to existing power value
                std powerlevel                  ; save new power value
                bra LD792                       ; go empty the flask and update things
LD783           clra                            ; new damage level = 0
                clrb
                bra LD790                       ; go set damage level and clean up flask
LD787           ldx powerlevel                  ; fetch player power level
                lda #$66                        ; roughly 0.8
                jsr applyscale                  ; go calculate 80% of player power level
                addd damagelevel                ; add that to the current damage level
LD790           std damagelevel                 ; save new damage level
LD792           ldb #$17                        ; type for "EMPTY"
                stb 9,u                         ; change flask to EMPTY
                clr 11,u                        ; mark flask as revealed
                playsoundimm $0c                ; play the flask sound
                updatestatus                    ; update status line to reflect changed flask state
                checkdamage                     ; check the damage level and recovery interval
                rts                             ; return to caller
LD7A0           clra                            ; flag for not showing creatures
                skip2                           ; skip over next instruction
```

```
LD7A2           lda #$ff                        ; flag for do show creatures
                sta showseer                    ; set creature display flag
                tst 11,u                        ; is flask revealed?
                bne LD7B6                       ; brif not - do nothing
                playsoundimm $0e                ; play the scroll sound
                clr hidestatus                  ; flag command processor to do a "restart"
                ldx #displayscroll              ; point to scroll display routine
                stx displayptr                  ; set the display handler
                updatedungeon                   ; update display with scroll
LD7B6           rts                             ; return to caller
cmd_zload       bsr LD7BC                       ; parse the file name
                dec loadsaveflag                ; flag ZLOAD
                rts                             ; return to caller
LD7BC           ldx #wordbuff                   ; get start address to set to $ff
                leau $20,x                      ; set $20 bytes
                setblock                        ; go clear block to $ff
                jmp LCB96                       ; go parse a word off command
cmd_zsave       bsr LD7BC                       ; parse the file name
                stx CBUFAD                      ; point buffer to file name
                ldd #$0f                        ;* set block type to header, length to 15
                std BLKTYP                      ;*
                inc loadsaveflag                ; flag ZSAVE
                rts                             ; return to caller
; Objects in backpack for demo game
startobjdemo    fcb 13                          ; iron sword
                fcb 15                          ; pine torch
                fcb 16                          ; leather shield
                fcb $ff                         ; end of list
; Objects in backpack for normal game
startobj        fcb 17                          ; wooden sword
                fcb 15                          ; pine torch
                fcb $ff                         ; end of list
; This is the list of routines that get scheduling entries by default.
LD7DC           fdb LD1EB                       ; keyboard input processing
                fdb LD1C2                       ; dungeon display update
                fdb LD1D5                       ; damage healing tick
                fdb LD19B                       ; tick down torch life
                fdb LD027                       ; add the "revenge" monsters for the current level
                fdb 0                           ; end of routine list
; cold start variable initializers
LD7E8           fcb 12
                fdb $103
                jmp swi2svc                     ; SWI2 handler
                jmp swisvc                      ; SWI handler
                jmp irqsvc                      ; NMI handler (why??)
                jmp irqsvc                      ; IRQ handler
                fcb $17
                fdb V202
                fcb $01                         ; V202 - apparently unused
                fdb $ffff                       ; allones - 16 bit all ones value, or -1
                fdb 128                         ; horizcent
                fdb 76                          ; vertcent
                fdb LD870                       ; screenvis - pointer to primary display screen info
                fdb LD876                       ; screendraw - pointer to secondary display screen info
                fdb demogame                    ; demoseqptr - pointer to demo game command sequence
                fdb objecttab                   ; objectfree - next free object entry
                fdb linebuff                    ; linebuffptr - the line input buffer pointer
                fcb 12,22                       ; playerloc - starting coordinates in maze (y, x)
                fdb $23                         ; carryweight - the weight of objects the player is carrying
                fdb $17a0                       ; powerlevel - player power level
                fcb $54
                fdb infoarea
                fdb $1000                       ; infoarea - text area starts at top of screen
                fdb $0260                       ; infoarea+2 - text area ends after 19 lines
                fdb 0                           ; infoarea+4 - text cursor position at top of screen
                fcb 0                           ; infoarea+6 - black background
                fcb $ff                         ; infoarea+7 - do not render on secondary screen
                fdb $2300                       ; statusarea - text area starts at row 19 on screen
                fdb $40                         ; statusarea+2 - text area goes for two lines
                fdb 0                           ; statusarea+4 - text cursor is at top of area
                fcb $ff                         ; statusarea+6 - background is white
                fcb 0                           ; statusarea+7 - do render on secondary screen
                fdb $2400                       ; commandarea - text area starts at row 20 on screen
                fdb $80                         ; commandarea+2 - text area goes for four lines
                fdb 0                           ; commandarea+4 - text cursor is at top of area
                fcb 0                           ; commandarea+6 - background is black
                fcb 0                           ; commandarea+7 - do render on secondary screen
                fcb 9,9,4,2,0,0,0,0,0,0,0,0     ; initial creature counts for level 1
                fcb 2,4,0,6,6,6,0,0,0,0,0,0     ; initial creature counts for level 2
                fcb 0,0,0,4,0,6,8,4,0,0,1,0     ; initial creature counts for level 3
                fcb 0,0,0,0,0,0,8,6,6,4,0,0     ; initial creature counts for level 4
                fcb 2,2,2,2,2,2,2,4,4,8,0,1     ; initial creature counts for level 5
                fcb 4
                fdb emptyhand+10
                fcb $04,$00,$00,$05             ; empty hand attack data
                fcb 0

; these tables are used for clearing and otherwise setting up the graphics screens
LD870           fdb $1000                       ; primary screen start address
                fdb $2300                       ; primary screen gfx area end address
                fdb $2046                       ; primary screen SAM register value
LD876           fdb $2800                       ; secondary screen start address
```

```
                fdb $3b00                       ; secondary screen gfx area end address
                fdb $20a6                       ; secondary screen SAM register value
LD87C           fdb $2300                       ; start address of status line on first screen
                fdb $2400                       ; end address of status line on first screen
                fdb 0                           ; dummy (SAM regster setting)
                fdb $3b00                       ; start address of status line on second screen
                fdb $3c00                       ; end address of status line on second screen
                fdb 0                           ; dummy (SAM register setting)
LD888           fdb $2400                       ; start address of command area on first screen
                fdb $2800                       ; end address of command area on first screen
                fdb 0                           ; dummy (SAM register setting)
                fdb $3c00                       ; start address of command area on second screen
                fdb $4000                       ; end address of command area on second screen
                fdb 0                           ; dummy (SAM register setting)

; This is the keyword table used for command parsing. Each keyword is stored in packed format.
; Each keyword is preceded by a value which indicates the object type. Where the object type is
; not relevant, that value will be zero. The value is shown in parentheses below.
kwlist_cmd      fcb 15                          ; 15 keywords in the command list
kw_attack       fcb $30,$03,$4a,$04,$6b         ; "ATTACK" keyword
                fcb $28,$06,$c4,$b4,$40         ; "CLIMB" keyword
                fcb $20,$09,$27,$c0             ; "DROP" keyword
kw_examine      fcb $38,$0b,$80,$b5,$2e,$28     ; "EXAMINE" keyword
                fcb $18,$0e,$5a,$00             ; "GET" keyword
                fcb $30,$12,$e1,$85,$d4         ; "INCANT" keyword
kw_look         fcb $20,$18,$f7,$ac             ; "LOOK" keyword
kw_move         fcb $20,$1A,$fb,$14             ; "MOVE" keyword
kw_pull         fcb $20,$21,$56,$30             ; "PULL" keyword
                fcb $30,$24,$5b,$14,$2c         ; "REVEAL" keyword
                fcb $20,$27,$47,$dc             ; "STOW" keyword
kw_turn         fcb $20,$29,$59,$38             ; "TURN" keyword
kw_use          fcb $18,$2b,$32,$80             ; "USE" keyword
                fcb $28,$34,$c7,$84,$80         ; "ZLOAD" keyword
                fcb $28,$35,$30,$d8,$a0         ; "ZSAVE" keyword
kwlist_dir      fcb 6                           ; 6 keywords in direction list
kw_left         fcb $20,$18,$53,$50             ; "LEFT" keyword
kw_right        fcb $28,$24,$93,$a2,$80         ; "RIGHT" keyword
                fcb $20,$04,$11,$ac             ; "BACK" keyword
                fcb $30,$03,$27,$d5,$c4         ; "AROUND" keyword
                fcb $10,$2b,$00                 ; "UP" keyword
                fcb $20,$08,$fb,$b8             ; "DOWN" keyword
kwlist_adj      fcb 25                          ; 25 keywords in the misc keywords list
kw_supreme      fcb $38,$67,$58,$48,$ad,$28     ; "SUPREME" keyword (1)
                fcb $28,$54,$fa,$b0,$a0         ; "JOULE" keyword (1)
                fcb $31,$0a,$cb,$26,$68         ; "ELVISH" keyword (4)
                fcb $38,$da,$9a,$22,$49,$60     ; "MITHRIL" keyword (3)
                fcb $20,$a6,$52,$c8             ; "SEER" keyword (2)
                fcb $28,$28,$82,$de,$60         ; "THEWS" keyword (0)
                fcb $20,$64,$96,$94             ; "RIME" keyword (1)
                fcb $30,$ac,$99,$a5,$ee         ; "VISION" keyword (2)
                fcb $20,$02,$2c,$94             ; "ABYE" keyword (0)
                fcb $20,$10,$16,$14             ; "HALE" keyword (0)
                fcb $29,$66,$f6,$06,$40         ; "SOLAR" keyword (5)
                fcb $30,$c5,$27,$bb,$45         ; "BRONZE" keyword (3)
                fcb $30,$6d,$56,$0c,$2e         ; "VULCAN" keyword (1)
                fcb $21,$13,$27,$b8             ; "IRON" keyword (4)
                fcb $29,$59,$57,$06,$40         ; "LUNAR" keyword (5)
                fcb $21,$60,$97,$14             ; "PINE" keyword (5)
                fcb $38,$d8,$50,$d1,$05,$90     ; "LEATHER" keyword (3)
                fcb $31,$2e,$f7,$90,$ae         ; "WOODEN" keyword (4)
                fcb $28,$4c,$97,$05,$80         ; "FINAL" keyword (1)
                fcb $30,$4a,$e2,$c8,$f9         ; "ENERGY" keyword (1)
                fcb $18,$52,$32,$80             ; "ICE" keyword (1)
                fcb $20,$4c,$99,$14             ; "FIRE" keyword (1)
                fcb $20,$4e,$f6,$10             ; "GOLD" keyword (1)
                fcb $28,$0a,$d8,$53,$20         ; "EMPTY" keyword (0)
                fcb $21,$48,$50,$90             ; "DEAD" keyword (5)
kwlist_obj      fcb 6                           ; 6 object types in the following list
kw_flask        fcb $28,$0c,$c0,$cd,$60         ; "FLASK" keyword (0)
                fcb $20,$64,$97,$1c             ; "RING" keyword (1)
                fcb $30,$a6,$39,$3d,$8c         ; "SCROLL" keyword (2)
kw_shield       fcb $30,$e6,$84,$95,$84         ; "SHIELD" keyword (3)
kw_sword        fcb $29,$27,$77,$c8,$80         ; "SWORD" keyword (4)
kw_torch        fcb $29,$68,$f9,$0d,$00         ; "TORCH" keyword (5)
; The following is the sequence of commands used in the demo game
demogame        fcb 1                           ; EXAMINE
                fdb kw_examine
                fcb 3                           ; PULL RIGHT TORCH
                fdb kw_pull
                fdb kw_right
                fdb kw_torch
                fcb 2                           ; USE RIGHT
                fdb kw_use
                fdb kw_right
                fcb 1                           ; LOOK
                fdb kw_look
                fcb 1                           ; MOVE
                fdb kw_move
                fcb 3                           ; PULL LEFT SHIELD
                fdb kw_pull
                fdb kw_left
                fdb kw_shield
                fcb 3                           ; PULL RIGHT SWORD
                fdb kw_pull
                fdb kw_right
                fdb kw_sword
                fcb 1                           ; MOVE
                fdb kw_move
                fcb 1                           ; MOVE
                fdb kw_move
                fcb 2                           ; ATTACK RIGHT
                fdb kw_attack
                fdb kw_right
                fcb 2                           ; TURN RIGHT
                fdb kw_turn
                fdb kw_right
                fcb 1                           ; MOVE
                fdb kw_move
                fcb 1                           ; MOVE
                fdb kw_move
                fcb 1                           ; MOVE
                fdb kw_move
                fcb 2                           ; TURN RIGHT
                fdb kw_turn
                fdb kw_right
                fcb 1                           ; MOVE
                fdb kw_move
                fcb 1                           ; MOVE
                fdb kw_move
                fcb $ff
; jump table for commands
LD9D0           fdb cmd_attack                  ; ATTACK
                fdb cmd_climb                   ; CLIMB
                fdb cmd_drop                    ; DROP
                fdb cmd_examine                 ; EXAMINE
                fdb cmd_get                     ; GET
                fdb cmd_incant                  ; INCANT
                fdb cmd_look                    ; LOOK
                fdb cmd_move                    ; MOVE
                fdb cmd_pull                    ; PULL
                fdb cmd_reveal                  ; REVEAL
                fdb cmd_stow                    ; STOW
                fdb cmd_turn                    ; TURN
                fdb cmd_use                     ; USE
                fdb cmd_zload                   ; ZLOAD
                fdb cmd_zsave                   ; ZSAVE
; pointers to the image data for object types
LD9EE           fdb img_flask                   ; flask
```

```
                fdb img_ring                    ; ring
                fdb img_scroll                  ; scroll
                fdb img_shield                  ; shield
                fdb img_sword                   ; sword
                fdb img_torch                   ; torch

LD9FA           fcb $05,$01

LD9FC           fcb $0A,$19,$19,$0A
; This is the object data table. Each entry is four bytes as follows:
; 0     object type
; 1     reveal strength required
; 2     magical offense multiplier
; 3     physical offense multiplier
objspecs        fcb $01,$FF,$00,$05             ; supreme ring
                fcb $01,$AA,$00,$05             ; joule ring
                fcb $04,$96,$40,$40             ; elvish sword
                fcb $03,$8C,$0D,$1A             ; mithril shield
                fcb $02,$82,$00,$05             ; seer scroll
                fcb $00,$46,$00,$05             ; thews flask
                fcb $01,$34,$00,$05             ; rime ring
                fcb $02,$32,$00,$05             ; vision scroll
                fcb $00,$30,$00,$05             ; abye flask
                fcb $00,$28,$00,$05             ; hale flask
                fcb $05,$46,$00,$05             ; solar torch
                fcb $03,$19,$00,$1A             ; bronze shield
                fcb $01,$0D,$00,$05             ; vulcan ring
                fcb $04,$0D,$00,$28             ; iron sword
                fcb $05,$19,$00,$05             ; lunar torch
                fcb $05,$05,$00,$05             ; pine torch
                fcb $03,$05,$00,$0A             ; leather shield
                fcb $04,$05,$00,$10             ; wooden sword
                fcb $01,$00,$00,$00             ; final ring
                fcb $01,$00,$FF,$FF             ; energy ring
                fcb $01,$00,$FF,$FF             ; ice ring
                fcb $01,$00,$FF,$FF             ; fire ring
                fcb $01,$00,$00,$05             ; gold ring
                fcb $00,$00,$00,$05             ; empty flask
                fcb $05,$05,$00,$05             ; dead torch
; This table has additional object data including ring charges, etc, organized as follows:
; 0     object number
; 1     burn time (torch), charges (ring), magical defense (shield)
; 2     physical light (torch), physical defense (shield)
; 3     magical ight (torch)
objextraspecs   fcb $00,$03,$12,$00             ; supreme ring
                fcb $01,$03,$13,$00             ; joule ring
                fcb $03,$40,$40,$00             ; mithril shield
                fcb $06,$03,$14,$00             ; rime ring
                fcb $0A,$3C,$0D,$0B             ; solar torch
                fcb $0B,$60,$80,$00             ; bronze shield
                fcb $0C,$03,$15,$00             ; vulcan ring
                fcb $0E,$1E,$0A,$04             ; lunar torch
                fcb $0F,$0F,$07,$00             ; pine torch
                fcb $10,$6C,$80,$00             ; leather shield
                fcb $18,$00,$00,$00             ; dead torch
                fcb $FF                         ; end of table
; This is the table of objects to create for a game. Each entry corresponds to
; a single object type. The first nibble is the minimum level number on which it
; appears. The second nibble is the number of objects of that type to generate.
; Generation starts at the specified level and creates one object assigned to
; that level. Then it creates another assigned to the next level, and so on.
; If it gets to level 5, it will reset to the minimum level. It cycles like this
; until there are the specified number of objects in the entire game.
LDA91           fcb $41                         ; 1 supreme ring, level 5
                fcb $31                         ; 1 joule ring, level 4
                fcb $31                         ; 1 elvish sword, level 4
                fcb $32                         ; 1 mithril shield each, level 4 and 5
                fcb $23                         ; 1 seer scroll each, level 3-5
                fcb $23                         ; 1 thews flask each, level 3-5
                fcb $11                         ; 1 rime ring, level 2
                fcb $13                         ; 1 vision scrool each, level 2-4
                fcb $16                         ; 2 abye flask each, level 2-3; 1 abye flask each level 4-5
                fcb $14                         ; 1 hale flask each, level 2-5
                fcb $14                         ; 1 solar torch each, level 2-5
                fcb $16                         ; 2 bronze shield each, level 2-3; 1 bronze shield each, level
4-5
                fcb $01                         ; 1 vulcan ring, level 1
                fcb $04                         ; 1 iron sword each, level 1-4
                fcb $08                         ; 2 lunar torch each, level 1-3; 1 lunar torch each, level 4-5
                fcb $08                         ; 2 pine torch each, level 1-3; 1 pine torch each, level 4-5
                fcb $03                         ; 1 leather shield each, level 1-3
                fcb $04                         ; 1 wooden sword each, level 1-4
; pointers to creature images
LDAA3           fdb LDE26                       ; spider
                fdb LDFCA                       ; viper
                fdb LDD41                       ; club giant
                fdb LDE59                       ; blob
                fdb LDE82                       ; knight
                fdb LDD51                       ; axe giant
                fdb LDE3F                       ; scorpion
                fdb LDE9D                       ; shield knight
                fdb LDE07                       ; wraith
                fdb LDDA3                       ; galdrog
                fdb img_wizardgen               ; wizard's image
                fdb img_wizard                  ; wizard
; This is the creature data table. Each entry is 8 bytes organized as follows:
; 0,1   creature power level
; 2     creature magical attack strength
; 3     creature magical defense strength
; 4     creature physical attack strength
; 5     creature physical defense strength
; 6     creature scheduling speed (movement) (in tenths of a second)
; 7     creature scheduling speed (attack) (in tenths of a second)
LDABB           fcb $00,$20,$00,$FF,$80,$FF,$17,$0B ; spider
                fcb $00,$38,$00,$FF,$50,$80,$0F,$07 ; viper
                fcb $00,$C8,$00,$FF,$34,$C0,$1D,$17 ; club giant
                fcb $01,$30,$00,$FF,$60,$A7,$1F,$1F ; blob
                fcb $01,$F8,$00,$80,$60,$3C,$0D,$07 ; knight
                fcb $02,$C0,$00,$80,$80,$30,$11,$0D ; axe giant
                fcb $01,$90,$FF,$80,$FF,$80,$05,$04 ; scorpion
                fcb $03,$20,$00,$40,$FF,$08,$0D,$07 ; shield knight
                fcb $03,$20,$C0,$10,$C0,$08,$03,$03 ; wraith
                fcb $03,$E8,$FF,$05,$FF,$03,$04,$03 ; galdrog
                fcb $03,$E8,$FF,$06,$FF,$00,$0D,$07 ; wizard's image
                fcb $1F,$40,$FF,$06,$FF,$00,$0D,$07 ; wizard
; This is the text font - these values are in packed format
LDB1B           fcb $30,$00,$00,$00,$00         ; char code 0 - space
                fcb $31,$15,$18,$fe,$31         ; char code 1 - A
                fcb $37,$a3,$1f,$46,$3e         ; char code 2 - B
                fcb $33,$a3,$08,$42,$2e         ; char code 3 - C
                fcb $37,$a3,$18,$c6,$3e         ; char code 4 - D
                fcb $37,$e1,$0f,$42,$1f         ; char code 5 - E
                fcb $37,$e1,$0f,$42,$10         ; char code 6 - F
                fcb $33,$e3,$08,$4e,$2f         ; char code 7 - G
                fcb $34,$63,$1f,$c6,$31         ; char code 8 - H
                fcb $33,$88,$42,$10,$8e         ; char code 9 - I
                fcb $30,$42,$10,$86,$2e         ; char code 10 - J
                fcb $34,$65,$4c,$52,$51         ; char code 11 - K
                fcb $34,$21,$08,$42,$1f         ; char code 12 - L
                fcb $34,$77,$5a,$d6,$31         ; char code 13 - M
                fcb $34,$63,$9a,$ce,$31         ; char code 14 - N
                fcb $33,$a3,$18,$c6,$2e         ; char code 15 - O
                fcb $37,$a3,$1f,$42,$10         ; char code 16 - P
                fcb $33,$a3,$18,$d6,$4d         ; char code 17 - Q
                fcb $37,$a3,$1f,$52,$51         ; char code 18 - R
                fcb $33,$a3,$07,$06,$2e         ; char code 19 - S
                fcb $37,$ea,$42,$10,$84         ; char code 20 - T
                fcb $34,$63,$18,$c6,$2e         ; char code 21 - U
                fcb $34,$63,$15,$28,$84         ; char code 22 - V
                fcb $34,$63,$1a,$d7,$71         ; char code 23 - W
                fcb $34,$62,$a2,$2a,$31         ; char code 24 - X
                fcb $34,$62,$a2,$10,$84         ; char code 25 - Y
                fcb $37,$c2,$22,$22,$1f         ; char code 26 - Z
                fcb $31,$08,$42,$10,$04         ; char code 27 - !
                fcb $30,$00,$00,$00,$1f         ; char code 28 - underscore
                fcb $33,$a2,$13,$10,$04         ; char code 29 - ?
                fcb $30,$00,$00,$00,$04         ; char code 30 - .
; some special glyphs
LDBB6           fcb $00,$00,$01,$01,$00,$00,$00 ; char code 32 - left part of contracted heart
                fcb $00,$a0,$f0,$f0,$e0,$40,$00 ; char code 33 - right part of contracted heart
                fcb $00,$01,$03,$03,$01,$00,$00 ; char code 34 - left half of expanded heart
                fcb $00,$b0,$f8,$f8,$f0,$e0,$40 ; char code 35 - right part of expanded heart

; These two entries are related to sound generation.
LDBD2           fcb $00,$80,$00,$01,$00,$50,$00,$04     ; for the "wizard fade out" sound and the walk into
wall sound
LDBDA           fcb $00,$50,$00,$05                     ; for the create death sound

; This table is for rendering walls in specific directions. There is one entry each
; for left, right, and forward. Each entry has four pointers to graphics, for no door,
; physical door, magical door, and solid wall.
LDBDE           fcb 3
                fdb LDC4F
                fdb LDC6B
                fdb LDC9B
                fdb LDC33

                fcb 0
                fdb LDC6A
                fdb LDC8B
                fdb LDCA9
                fdb LDC45
                fcb 1
                fdb LDC5D
                fdb LDC7B
                fdb LDCA2
                fdb LDC3C
                fcb $ff

; image data for a shield
img_shield      fcb 134,172
                fcb 128,192
                fcb 122,186
                fcb 128,168
                fcb $fc
                fcb $3e,$04,$00
                fcb $fe
; image data for a torch
img_torch       fcb 118,60
                fcb $fc
                fcb $f7,$ff,$2a,$00
                fcb $fe
; image data for a sword
img_sword       fcb 114,80
                fcb 124,100
                fcb $ff
                fcb 118,82
                fcb 114,86
                fcb $fe

; image data for a flask
img_flask       fcb 110,162
                fcb $fc
                fcb $51,$0e,$b1,$00
                fcb $fe
; image data for a ring
img_ring        fcb 122,60
                fcb $fc
                fcb $11,$1f,$ff,$f1,$00
                fcb $fe
; image data for a scroll
img_scroll      fcb 118,194
                fcb $fc
                fcb $1f,$34,$f1,$dc,$00
                fcb $fe

; Creature around corner to the left indicator graphic
LDC33           fcb 16,27
                fcb 38,64
                fcb 114,64
                fcb 136,27
                fcb $fe
; Creature around corner to the right indicator graphic
LDC3C           fcb 16,229
                fcb 38,192
                fcb 114,192
                fcb 136,229
                fcb $fe
LDC45           fcb 38,64
                fcb 38,192
                fcb $ff
                fcb 114,64
                fcb 114,192
                fcb $fe
LDC4F           fcb 38,29
                fcb 38,64
                fcb 114,64
                fcb 114,27
                fcb $ff
                fcb 16,27
                fcb 38,64
                fcb $fe
LDC5D           fcb 38,229
                fcb 38,192
                fcb 114,192
                fcb 114,229
                fcb $ff
                fcb 16,229
                fcb 38,192
LDC6A           fcb $fe
LDC6B           fcb 128,40
                fcb 65,40
                fcb 68,56
                fcb 119,56
                fcb $ff
                fcb 92,48
                fcb 93,52
                fcb $fd
                fdb LDC33
LDC7B           fcb 128,216
                fcb 65,216
                fcb 68,200
                fcb 119,200
                fcb $ff
                fcb 92,208
                fcb 93,204
                fcb $fd
                fdb LDC3C
LDC8B           fcb 114,108
                fcb 67,108
                fcb 67,148
                fcb 114,148
                fcb $ff
                fcb 94,126
                fcb 94,130
                fcb $fd
                fdb LDC45
```

```
LDC9B           fcb 128,40
                fcb 66,50
                fcb 117,58
                fcb $fe
LDCA2           fcb 128,216
                fcb 66,206
                fcb 117,198
                fcb $fe
LDCA9           fcb 113,108
                fcb 67,128
                fcb 114,148
                fcb $fe
LDCB0           fcb 100,28
                fcb $fc
                fcb $44,$2e,$42,$4c,$00
                fcb $fe
LDCB9           fcb 100,228
                fcb $fc
                fcb $4c,$22,$4e,$44,$00
                fcb $fe
; Table of pointers to hole/ladder graphics
LDCC2           fdb LDD0E
                fdb LDCCA
                fdb LDD2A
                fdb LDCD0
LDCCA           fcb $fb
                fdb LDCD6
                fcb $fd
                fdb LDD0E
LDCD0           fcb $fb
                fdb LDCD6
                fcb $fd
                fdb LDD2A
LDCD6           fcb 24,116
                fcb 128,116
                fcb $ff
                fcb 24,140
                fcb 128,140
                fcb $ff
                fcb 28,116
                fcb 28,140
                fcb $ff
                fcb 40,116
                fcb 40,140
                fcb $ff
                fcb 52,116
                fcb 52,140
                fcb $ff
                fcb 64,116
                fcb 64,140
                fcb $ff
                fcb 76,116
                fcb 76,140
                fcb $ff
                fcb 88,116
                fcb 88,140
                fcb $ff
                fcb 100,116
                fcb 100,140
                fcb $ff
                fcb 112,116
                fcb 112,140
                fcb $ff
                fcb 123,116
                fcb 123,140
                fcb $ff
                fcb $fa
LDD0E           fcb 34,100
                fcb 24,92
                fcb 24,164
                fcb 34,156
                fcb 34,100
                fcb 24,100
                fcb $ff
                fcb 34,156
                fcb 24,156
                fcb $ff
                fcb 28,47
                fcb 28,96
                fcb $ff
                fcb 28,161
                fcb 28,210
                fcb $fe
LDD2A           fcb 118,100
                fcb 128,92
                fcb 128,164
                fcb 118,156
                fcb 118,100
                fcb 128,100
                fcb $ff
                fcb 118,156
                fcb 128,156
                fcb $ff
LDD3C           fcb 28,47
                fcb 28,210
                fcb $fe
LDD41           fcb 104,98
                fcb $fc
                fcb $d7,$d4,$14,$12,$30,$1d,$0d,$fd
                fcb $29,$00
                fcb $fd
                fdb LDD62
LDD51           fcb 104,98
                fcb 94,124
                fcb 96,126
                fcb 106,100
                fcb $ff
                fcb 102,132
                fcb 92,114
                fcb 102,118
                fcb 110,114
LDD62           fcb 102,132
                fcb $fc
                fcb $02,$56,$56,$17,$ee,$02,$ea,$bb
                fcb $bb,$ea,$ea,$00
                fcb 78,92
                fcb $fc
                fcb $c2,$51,$3e,$cf,$fc,$42,$13,$00
                fcb 106,90
                fcb $fc
                fcb $1e,$11,$f3,$62,$39,$e2,$0c,$e4
                fcb $8a,$e2,$00
                fcb 86,84
                fcb $fc
                fcb $54,$65,$2e,$ca,$ba,$a1,$d4,$ee
                fcb $12,$d2,$13,$e1,$20,$f6,$24,$72
                fcb $58,$ee,$c5,$be,$00
                fcb $fe
LDDA3           fcb 80,124
                fcb 94,114
                fcb 110,120
                fcb 132,112
                fcb 104,78
                fcb 132,48
                fcb 68,72
                fcb 84,32
                fcb 22,88
                fcb 52,114
                fcb 92,128
                fcb 52,142
                fcb 22,168
                fcb 88,224
                fcb 68,184
                fcb 132,208
                fcb 112,178
                fcb 132,144
                fcb 110,136
                fcb 94,142
                fcb 80,132
                fcb $ff
                fcb 132,112
                fcb $fc
                fcb $c5,$92,$be,$c3,$43,$5e,$72,$45
                fcb $00
                fcb 82,122
                fcb $fc
                fcb $78,$e9,$8d,$ec,$33,$0c,$24,$72
                fcb $47,$e7,$00
                fcb 22,168
                fcb $fc
                fcb $2d,$c2,$3d,$30,$4b,$4b,$ed,$b2
                fcb $9d,$71,$3d,$dd,$91,$7d,$52,$63
                fcb $a3,$2d,$ed,$2d,$cb,$cb,$d0,$dd
                fcb $42,$ed,$00
                fcb $fe
LDE07           fcb 62,68
                fcb 68,88
                fcb 56,100
                fcb $ff
                fcb 74,90
                fcb 70,74
                fcb $fc
                fcb $33,$f5,$f5,$c1,$5a,$62,$0e,$00
                fcb 100,80
                fcb $fc
                fcb $b3,$17,$34,$eb,$0a,$3d,$00
                fcb $fe
LDE26           fcb 124,160
                fcb $fc
                fcb $c2,$22,$e4,$24,$2c,$ec,$04,$04
                fcb $e2,$42,$00
                fcb 124,168
                fcb $fc
                fcb $c1,$21,$12,$f2,$e1,$41,$00
                fcb $fe
LDE3F           fcb 112,74
                fcb $fc
                fcb $e0,$ee,$2c,$42,$14,$14,$20,$0c
                fcb $cc,$22,$0c,$22,$00
                fcb 124,90
                fcb $fc
                fcb $e0,$0c,$2c,$20,$04,$00
                fcb $fe
LDE59           fcb 82,130
                fcb $fc
                fcb $28,$7d,$5f,$50,$5b,$f5,$2f,$d5
                fcb $17,$17,$f3,$22,$e1,$14,$dd,$8f
                fcb $8d,$db,$ec,$00
                fcb 86,130
                fcb $fc
                fcb $33,$31,$1b,$91,$3b,$5f,$f5,$00
                fcb 108,116
                fcb 114,118
                fcb 120,144
                fcb $fe
LDE82           fcb 34,124
                fcb $fc
                fcb $04,$1f,$0e,$ff,$00
                fcb 80,142
                fcb 64,136
                fcb 46,146
                fcb 64,156
                fcb 82,140
                fcb 76,136
                fcb 64,146
                fcb 58,140
                fcb $fd
                fdb LDEB3
LDE9D           fcb 30,126
                fcb $fc
                fcb $50,$0f,$e0,$00
                fcb 44,150
                fcb 52,166
                fcb 76,164
                fcb 92,150
                fcb 76,136
                fcb 52,134
                fcb 44,150
                fcb $ff
LDEB3           fcb 80,140
```

```
                fcb 128,152
                fcb 132,160
                fcb 132,144
                fcb 126,144
                fcb 84,130
                fcb $ff
                fcb 84,126
                fcb 126,110
                fcb 132,110
                fcb 132,92
                fcb 128,102
                fcb 80,116
                fcb $ff
                fcb 80,140
                fcb $fc
                fcb $3a,$d9,$83,$de,$ad,$e6,$a1,$e2
                fcb $22,$61,$26,$ea,$20,$3d,$dd,$e0
                fcb $00
                fcb 52,128
                fcb 20,128
                fcb $fc
                fcb $0e,$21,$02,$e1,$0e,$00
                fcb 74,102
                fcb $fc
                fcb $e0,$02,$d0,$08,$30,$02,$20,$01
                fcb $30,$02,$d0,$01,$87,$00
                fcb 46,110
                fcb 64,102
                fcb 64,100
                fcb 30,102
                fcb 20,98
                fcb 30,94
                fcb 64,96
                fcb 64,98
                fcb 20,98
                fcb $FE
; Image for the Wizard
img_wizard      fcb 46,98
                fcb $fc
                fcb $21,$2f,$2d,$fd,$ce,$c2,$f2,$12
                fcb $0f,$1e,$3f,$21,$12,$e3,$e0,$00
                fcb 104,154
                fcb $fc
                fcb $21,$2f,$2d,$fd,$ce,$c2,$f2,$12
                fcb $0f,$1e,$3f,$22,$12,$e2,$e0,$00
                fcb $fd
                fdb img_wizardgen
; Image for the "good" wizard
img_goodwiz     fcb 40,86
                fcb 64,92
                fcb 42,100
                fcb 54,82
                fcb 56,104
                fcb 40,86
                fcb $ff
                fcb 66,140
                fcb $fc
                fcb $70,$ad,$35,$1b,$b3,$00
                fcb 96,146
                fcb 120,148
                fcb 100,136
                fcb 106,154
                fcb 116,138
                fcb 96,146
                fcb $ff
                fcb 80,116
                fcb $fc
                fcb $53,$ec,$e4,$4d,$b0,$00
img_wizardgen   fcb 64,124
                fcb $fc
                fcb $4e,$c0,$7b,$9c,$d4,$e4,$e1,$e1
                fcb $dd,$1c,$96,$03,$00
                fcb 28,130
                fcb $fc
                fcb $03,$45,$71,$da,$1e,$11,$e1,$00
                fcb 48,134
                fcb 54,142
                fcb 116,164
                fcb 132,132
                fcb 130,118
                fcb 120,94
                fcb 90,110
                fcb 132,132
                fcb 72,106
                fcb $ff
                fcb 64,102
                fcb $fc
                fcb $1f,$bd,$f1,$53,$00
                fcb 66,102
                fcb $fc
                fcb $1e,$32,$11,$73,$00
                fcb 88,112
```

```
                    fcb 72,120
                    fcb $ff
                    fcb 62,132
                    fcb 20,128
                    fcb 52,122
                    fcb 64,122
                    fcb 60,124
                    fcb 114,128
                    fcb 80,130
                    fcb 68,130
                    fcb 62,132
                    fcb $ff
                    fcb 40,130
                    fcb $fc
                    fcb $ff,$1e,$11,$f2,$3f,$20,$0f,$c0
                    fcb $ff,$31,$00
                    fcb $fe
LDFCA               fcb 132,130
                    fcb 112,122
                    fcb 92,124
                    fcb 94,126
                    fcb 94,130
                    fcb 92,132
                    fcb 112,130
                    fcb 128,140
                    fcb 132,136
                    fcb 132,114
                    fcb 120,108
                    fcb 106,118
                    fcb 120,112
                    fcb 124,116
                    fcb 124,126
                    fcb $ff
                    fcb 100,120
                    fcb $fc
                    fcb $e0,$e2,$ee,$e0,$f1,$22,$ee,$06
                    fcb $2e,$e2,$11,$20,$2e,$22,$20,$00
                    fcb $fe
                    fcc 'KSK'
```