```
* UNATRON main loop
        SETDP   $24     Set page "0" to $24
ADDCHQ  EQU     $2FA2   Subroutine for adding another character to the character list or "c-list".
ANTISH  EQU     $3088   Subroutine for erasing a char. from the screen at it's current location.
NEWLOC  EQU     $3074   Subroutine for computing char's new loc from current loc and char's vector.
NXTSET  EQU     $3017   Subroutine for moving cursor from shape data when writing shape.
OKMOV   EQU     $2FE5   Subroutine for checking to see if proposed new loc for char is already occupied.
REALCO  EQU     $2FCB   Subroutine for translating screen loc into memory loc and bit set.
VMULT4  EQU     $30FA   Subroutine for multiplying vector in VOUT by two.
WRTSHP  EQU     $3088   Subroutine for writing shape - shape addr in STSH, real loc in RLCC and RBIT.
BCVYSH  EQU     $2F58   Subroutine for building restore shape for a write-restore operation.
DWNVEC  EQU     $2EC1   Subroutine for generating vector from char to AIM loc on screen.
NEWVEC  EQU     $2ED4   Subroutine for generating vector from char to players character.
RNDVEC  EQU     $2F15   Subroutine for generating a random vector.
SHPADR  EQU     $2EB5   Subroutine for looking a shape's addr from the shape table.
KBDWAI  EQU     $2E71   Subroutine cycles and waits for fire button to be pushed.
CDISP   EQU     $2E88   Subroutine displays the number of player's characters left.
TALLY   EQU     $2CC4   Subroutine displays scoring.
CREAC   EQU     $245A   # of chain reactions in this round.
CCMH    EQU     $2459   # of computer hits in this round.
YURH    EQU     $2458   # of player's hits in this round.
TCREAC  EQU     $2456   Total number of chain reactions since start of game.
TCOMH   EQU     $2454   Total number of computer hits since start of game.
TYURH   EQU     $2452   Total number of player's hits since start of game.
ATRCT   EQU     $2490   Parameter controls strength of CURC's attraction to AIM location.
MENL    EQU     $2451   # of player's characters left in game.
EWAI    EQU     $2450   Wait counter to slow game down near the end of round.
PSC2    EQU     $2447   LSB of proposed screen location.
PSCR    EQU     $2446   Screen coordinate, real value of a point on screen; = 128 * Y + X.
RBIT    EQU     $2443   Bit set; selects pixel 1,2,3,or 4 in a byte defined by RLOC.
RLOC    EQU     $2444   Actual byte location of a pixel on the screen.
STSH    EQU     $243E   Where addr of the start of a shape is stored after calling SHPADR.
TMP2    EQU     $2431   Work storage.
NUMN    EQU     $241D   * Number of neutrons released when a mine is exploded.
TLOC    EQU     $2441   Working temporary for variable RLCC.
TBIT    EQU     $2440   Working temporary for variable RBIT.
VOU1    EQU     $2430   LSB of vector generated by vector generating routines.
VOUT    EQU     $243C   Vector generated by vector generating routines.
PSHP    EQU     $2438   Proposed Shape; where # of shape to be ADDCHQed is stored.
TMP1    EQU     $2432   Work storage.
CURC    EQU     $243A   Shape # of atom currently reactive.
RAWY    EQU     $2439   # non-zero value here will cause CURC to run away from player's char.
TEMX    EQU     $2433   Temp storage usually used for X register.
TVEC    EQU     $2437   Temp vector storage.
SVEC    EQU     $2435   Temp vector storage.
RND1    EQU     $242F   Working storage.
TMP3    EQU     $2430   Working storage.
STBO    EQU     $2402   * Addr of start of screen borders layout.
RND2    EQU     $242E   Working storage.
RND3    EQU     $242D   Working storage.
RND4    EQU     $242C   Working storage.
RND5    EQU     $242B   Working storage.
MAXH    EQU     $241A   * Max # of holemakers to appear on the screen this round.
NUMH    EQU     $242A   Number of holemakers currently on the screen.
NUMM    EQU     $2429   Number of mines currently on the screen.
MAXM    EQU     $2415   * Max # of mines that can appear on screen this round.
```

```
MANST   EQU     $2404   * Screen loc of where player's character starts.
SHP1ST  EQU     $2406   * Screen loc of where first atom starts
SETPTR  EQU     $2400   * Addr of next set of overlaid data.
MSCR    EQU     $2413   * Screen loc of where mines appear.
MBEFH   EQU     $2419   * Number of mines that must appear before a hole appears.
MINCH   EQU     $2416   * Relative chance of a mine appearing.
MINSPL  EQU     $2417   * The shortest number of cycles a mine will wait before chasing player.
MINSPH  EQU     $2418   * The longest number of cycles a mine will wait before chasing player.
MINR    EQU     $244F   Real memory addr of where a mine will appear.
MINB    EQU     $244E   Bit set of where a mine will appear.
GCH     EQU     $2412   * Relative chance of computer guns firing.
G2L     EQU     $2411   * Length of shot originating from gun #2.
G2V     EQU     $240F   * Vector for shot originating from gun #2.
G2S     EQU     $240D   * Screen loc of where gun #2 appears.
G2R     EQU     $244C   Real memory addr of where gun #2 appears.
G2B     EQU     $2448   Bit set of where gun #2 appears.
G1L     EQU     $240C   * Length of shot originating from gun #1.
G1V     EQU     $240A   * Vector for shot originating from gun #1.
G1S     EQU     $2408   * Screen loc of where gun #1 appears.
G1R     EQU     $2449   Real memory addr of where gun #1 appears.
G1B     EQU     $2448   Bit set of where gun #1 appears.
        ORG     $2500   Main routine starts here.
START   LDA     #$24    Value for direct page register.
        TFR     A,DP    Set direct page register.
        STA     65478   "0" setting video pages
        STA     65481   "1"
        STA     65482   "0"
        STA     65485   "1"
        STA     65487   "1"
        STA     65488   "0"
        STA     65472   "0" setting video mode G3C.
        STA     65474   "C"
        STA     65477   "1"
        LDA     #255    Value for video control register - gives black background and one of two color sets.
        STA     65314   Poke the value into the video control register. Video parameters are not reset from here on in.
        LDD     #$2200  Hard codeo address of the first set of data to be overlaid at from $2400 to $2429
        STD     SETPTR  Store the $2200 so that the first set of overlaid data is read starting from that point.
        LDS     #$7FF   Going to move the hardware stack out of the way. No data has been stored below $800.
        LDA     #C4     Give the player four characters to spend. This will be incremented to five as soon as score is put up.
        STA     MENL    Store it.
        LDD     #0      Zero out D register and use it to zero out point totals.
        STD     TYURH   Player's (your) total number of hits is assigned C.
        STD     TCOMH   Computer's total number of hits is assigned 0.
        STD     TCREAC  Total number of chain reactions is assigned 0.
        CLR     YURH    Clear player's hits for this round.
        CLR     CREAC   Clear number of chain reactions for this round.
        CLR     CCMH    Clear number of computer's hits for this round.
*
* One time initialization of the game has been completed.
* Now will retrieve $29 bytes of overlaid data starting at SETPTR.
* Data is written from $2400 to $2428. Note $24 has been set to
* page zero in the direct page register. Data in the $29 bytes
* controls the difficulty of this round, screen layout etc.
*
ZSTART  LDX     SETPTR  Start of a round: Get addr of next set of overlaid data.
        CLRA            There are $29 bytes (some unused) of overlaic data. The A register is going to be used to count to $29.
```

```
        LDY     #$24CC  Where overlaid data will go. The $29 bytes includes all parameters which change the game from round to roun
FM1     LDB     ,X+     Get a byte of the data.
        STB     ,Y+     Put the byte down.
        INCA            Increment the counter.
        CMPA    #$29    Have all $29 bytes been transferred?
        BEQ     FM2     If so, branch out of this loop.
        BRA     FM1     A register still not equal to $29, continue loop.
*
* The character c-list starts at 12547 and extencs to 13311.
* Video ram runs from 13312 to 16383. In one big sweep both
* will be initialized to zeros.
*
FM2     LDX     #12547  This is the addr of the start of the character c-list. The display ram is adjacent to the c-list. Clear bot
C200    CMPX    #16383  Check to see if the end of the video ram has been reachec.
        BGE     C201    If so, exit loop. Character c-list and video ram are cleared.
        CLR     ,X+     Zero out byte, X points to next byte.
        BRA     C200    Continue clearing c-list and video ram.
*
C201    JSR     TALLY   Now things are happening! Put up scoreboard.
        JSR     KBDWAI  What for fire button to be pushed.
*
* The first atom and the player are now set up. The screen locations
* where both will start are extracted from the data overlaid above
* and translated into real video ram locations and bit sets.
* The 1st atom and the player are added to the character c-list.
* Note the player's character is done first. This assures the
* player will always hold the first position in the c-list.
*
        LDD     MANST   Get player's starting screen location.
        STD     PSCR    Store it as a proposed screen location so REALCO can pick it up.
        JSR     REALCO  Change screen loc into a video ram address and bit set (pixel number). These are returned to RBIT and RLCC.
        LDA     #C6     This is the shape the player will start out with. It looks like this; V .
        STA     PSHP    Store it as the proposed shape so ADDCHC can pick it up.
        JSR     ADDCHC  Add player to c-list w/shape = C6, screen loc = PSCR, real loc = RLOC, and bit set = RBIT.
        LDD     SHP1ST  Get screen address of where atom #1 starts.
        STD     PSCR    Store it as proposed screen location.
        JSR     REALCO  Translate into real coordinates.
        LDA     #18     Shape #18 is the first atom in it's normal (as opposed to wobble) state.
        STA     PSHP    Store it as proposed shape.
        STA     CURC    Store it as the current character - the one the player is chasing, the one that can fission.
        JSR     ADDCHC  Add the character to the c-list.
*
* For the rest of the round the computer guns will be firing and
* mines will appear. The screen positions where these originate
* are read from the overlaid data section, translated into video
* ram locations and bit sets and stored so they can be fetched
* whenever a mine is to be born or a gun is to fire. Calculating
* this information once at the beginning of a round saves time later.
*
        LDD     MSCR    Get the address of the screen location where the mines are born.
        STD     PSCR    Store it as the proposed screen location.
        JSR     REALCO  Translate the screen location into a video ram location and bit set.
        LDD     RLOC    Get the video ram location.
        STD     MINR    Store it here. It will stay here and need not be recalculated whenever a mine is born.
        LDA     RBIT    Get the bit set just calculated.
        STA     MINB    Store it so it need not be recalculated again this round.
```

```
        LDD     G1S     Get the screen location where gun #1 appears.
        STD     PSCR    Store as proposed location.
        JSR     REALCO  Translate the screen location into a video ram location and bit set.
        LDD     RLOC    Get the video ram location.
        STD     G1R     Store it so it need not be recalculated this round.
        LDA     RBIT    Get the bit set.
        STA     G1B     Store it so it need not be recalculated.
        LDD     G2S     Get the screen location where gun #2 will appear.
        STD     PSCR    Store it as proposed location so REALCO can get it.
        JSR     REALCO  Translate to real coordinates.
        LDD     RLOC    Get the video ram address just generated.
        STD     G2R     Store it so it need not be recalculated.
        LDA     RBIT    Get the bit set.
        STA     G2B     Store it so it need not be recalculated.
*
        CLR     EWAI    Clear the end of round wait counter so that the game runs at normal speed.
        CLR     NUMH    Clear the number of holemakers counter (because there are none of course).
        LDA     #C7     An attraction parameter. The lower the value, the harder CURC will try to reach AIM, the screen location.
        STA     ATRCT   #07 is a moderate value, it is lowered later so that the end of the round isn't spent chasing 3 or 4 dots.
*
* The round has been set up. All actions from here in take place inside a given round.
*
RESTAR  CLR     NUMM    Every time the player is hit by a mine, or whenever a new round starts, mines are deleted from the c-list.
        LDX     #13312  Load X register with the start of the video ram. The screen is going to be cleared.
C2C2    CMPX    #16383  Check to see if the screen has been cleared.
        BGT     XXX     If it has, branch out of loop.
        CLR     ,X+     Clear a byte of video ram and increment X.
        BRA     C2C2    Continue with the loop.
*
* The border layout will now be drawn on the screen.
* The layout is made up of two shapes, #108 and #110.
* In the first two screens shape #108 is a horizontal brick and
* #110 is a vertical brick. The location STBO points to the start
* of a list of screen locations where the shapes are to be drawn.
* Shape #108 is drawn at all addresses plucked from the list
* until a negative address is encountered. Shape #110 is then drawn
* at the next bunch of addresses plucked from the list until
* a second negative number is encountered.
*
XXX     LDA     #108    The screen layout is made of two shapes. #108 is the first. The shape's address must be looked up.
        JSR     SHPADR  Shape number in A, address of the shape is returned to STSH.
        LDX     STBO    The screen address for each piece of the layout (borders) is in a table pointed to by STBO.
XX2     LDD     ,X++    Get two bytes from the table. Increment X two bytes.
        BLT     XX3     If the word loaded into D is < 0, this signals that all occurances (if any) of #108 have been seen. Branch.
        STD     PSCR    Store the bytes as a proposed screen address.
        JSR     REALCO  Translate the address into real coordinates.
        JSR     WRTSHP  Write the shape pointed to by STSH in video ram at RLOC with bit set RBIT.
        BRA     XX2     Continue stepping through the table.
XX3     LDA     #11C    This is the second shape in the screen layout.
        JSR     SHPADR  The shape addr is looked up repetitively. Wasteful, but time is cheap at this point.
        LDD     ,X++    Get the screen addr of the shape to be put up. X points to next shape.
        BLT     XXX4    If the screen addr loaded into D < 0 (i.e. end of data) exit loop.
        STD     PSCR    Store the addr as proposed screen location.
        JSR     REALCO  Translate it into real coordinates.
        JSR     WRTSHP  Write shape pointed to by STSH at RLOC with bit set RBIT.
        BRA     XX3     Continue looping through layout table.
```

```
*
* The mines and chain reaction neutrons are deleted from the screen.
* This saves the player from repeatedly being attacked by the same
* mine or from losing too many points to chain reactions. If this
* is the first pass through this section this round then there
* wont be any mines or neutrons in the c-list anyway. If is not
* the first pass through, the player has just been hit by a mine.
*
XXX4    LDX     #12547  Set X to point to start of c-list. Will loop through and delete mines and neutrons.
XXX5    CMPX    #13312  Check to see if at the end of the character c-list.
        BGE     LOOP    If so branch out of this loop to main LOOP.
        LDA     ,X      Get the shape number of the character from the c-list.
        CMPA    #42     Check to see if its a neutron from chain reaction.
        BEQ     XXX7    If it is, branch below where it'll be deleted.
        CMPA    #106    Check to see if character from c-list is a mine.
        BEQ     XXX7    If it is, branch below where it'll be deleted.
        BRA     XXX6    It is neither mine nor neutron. Branch below but skip the deletion part of the deal.
XXX7    CLR     ,X      The mine or neutron is deleted.
XXX6    LEAX    9,X     The X register is increased by 9 to point to the next character in the c-list.
        BRA     XXX5    Branch back up and continue looking for mines and neutrons.
*
* This completes the screen setup. The rest is normal operation of the game.
*
* This is the main loop of the game.
* First we will check the player's joystick pots and decide
* 1) What vector to give the player's character (49 possibilities).
* 2) What vector the player's shot will take if fired.
* 3) What the player's character will look like based on the
*    direction it is moving.
* The player can move at three speeds in a given direction. The speed
* is determined by how displaced the joystick is from the center.
* The player's shot vector is also prepared. It is twice the player's
* vector when the player's vector is non-zero. Otherwise it is left
* unmodified.
*
LOOP    JSR     [$A00A] Sample the joystick pots with ROM routine whose address is at $A00A.
        LDX     #12547  Load the X register with the start of the c-list. Player's character ALWAYS occupies the first position.
        CLR     $FF20   The d/a converter is at $FF20. It has just been used in sampling the joysticks and must be cleared for soun
        LDA     #$BC    Load A with value for routine output from d/a converter to TV sound modulator.
        STA     $FF23   Store it at this PIA.
        LDD     #0      The player's vector and shape and shot vector are now going to be decided.
        STD     TVEC    Clear temporary vector location.
        STD     VCUT    Clear vector location.
        STA     PSHP    Set proposed shape # to 0, values will be added to this to decide the final shape #.
        LDA     $C15B   Get vertical joytsick reading. Check for upward movement first. There are three speeds each direction.
        CMPA    #18     Up slowly,medium or fast - all less than or equal to 18 on joystick.
        BGT     C70     If greater than 18, check for downward movement indicated by joystick.
        LDB     #06     Shape for player's character facing up.
        STB     PSHP    Store as proposed shape.
        CMPA    #6      Joystick value of < 6 incicates quick upward movement.
        BGE     UC1     Value greater than or equal - check for medium speed, branch
        LDD     #$FF0C  It is upward quickly. Vector = $FF00 = -256.
        STD     TVEC    Store as temporary vector for player's character.
        LSLA            Vector for shot always twice player's character speed. Multiply by two.
        STD     VCUT    Store as vector for player's shot.
        BRA     C71     Branch to check horizontal movement.
```

```
U01     CMPA    #12     Medium speed indicated by a value less than 12 and greater than 5.
        BGE     LC2     If value is greater than or equal to 12 slow speed must be called for. Branch.
        LDD     #$FF8C  Vector for medium speed upward = FF80 = -128.
        STD     TVEC    Store as temporary vector for player's character.
        CLRB            Preparing vector for shot = twice player's speed. Clearing B makes value = $FF00 = -256.
        STD     VOUT    Store as vector for player's shot.
        BRA     C71     Branch to check for horizontal motion.
U02     LDD     #$FF8C  Slow speed.This will be vector for shot = -128.
        STD     VCUT    Store as vector for player's shot.
        INC     RND3    This location used as an odd/even counter for slow player vectors.
        LDA     RND3    Every other cycle player will move upward. Effectively half speed.
        ANDA    #C1     If non-zero result from this give player vector of -128, else leave = 0.
        BEQ     C71     Result was zero, branch.
        LDD     #$FF80  Vector equal to -128.
        STD     TVEC    Store as temporary vector for player's character.
        BRA     C71     Go do horizontal component.
C70     CMPA    #45     If ended up here, there was no upward movement given to player. Check for downward.
        BLE     C71     If joystick value < 45, no downward movement called for. Go check for horizontal.
        LDB     #12     There is downward movement. 12 is shape # for player's character facing downward.
        STB     PSHP    Store the 12 as proposed shape number.
        CMPA    #57     A value greater than 57 calls for fast downward movement.
        BLE     UC3     If value is less than or equal, branch to check for medium speed.
        LDD     #256    Vector for fast downward movement.
        STD     TVEC    Store as temporary vector for player's character.
        LSLA            Multiply by two so that shot moves at 512.
        STD     VCUT    Store shot's vector as vector.
        BRA     C71     Branch to check for horizontal movement.
U03     CMPA    #51     Check for medium speed.
        BLT     UC4     A value less than 51 calls for slow speed. Branch.
        LDD     #128    Vector for medium speed.
        STD     TVEC    Store as temporary vector for player's character.
        LDD     #256    Vector for shot = twice player's speed.
        STD     VCUT    Store as vector for player's shot.
        BRA     C71     Branch to check for horizontal movement.
UC4     LDD     #128    Slow speed. 128 is vector for shot.
        STD     VCUT    Store as vector for player's shot.
        INC     RND3    Increment odd/even counter.
        LDA     RND3    Get odd/even counter.
        ANDA    #C1     Non-zero result here and character moves down with vector of 128, else no movement downward.
        BEQ     C71     If result is zero, go check for horizontal movement.
        LDD     #128    Result not = C. Give player vector for downward movement.
        STD     TVEC    Store as vector for player's character.
C71     LDA     $D15A   Horizontal movement section. Get result for joystick sample.
        CMPA    #18     A value greater than 18 indicates no leftward movement.
        BGE     C72     If no leftward movement, branch to check for rightward movement.
        INC     PSHP    Adding two to player shape. If there was no vertical movement, PSHP will equal 2.
        INC     PSHP    If there was upward movement, PSHP will=8. If downward PSHP will=14. Takes care of diagonal shapes.
        CMPA    #C6     Check to see if fast leftword movement is called for.
        BGE     UC5     No, then check for medium speed.
        LDD     TVEC    Fast. Get player's vector generated so far.
        SUBD    #C2     Add two pixel leftward displacement.
        STD     TVEC    Store as temporary vector for player's character.
        LDD     VCUT    Get player's shot vector generated so far.
        SUBD    #C4     Give twice player's leftward displacement.
        STD     VCUT    Store as vector for player's shot.
        BRA     C73     Exit player vector generating section.
```

```
U05     CMPA    #12     Check for medium speed.
        BGT     U06     No, check for slow speed.
        LDD     TVEC    Medium speed. Getplayer's vector generated so far.
        SUBD    #01     Give one pixel leftward displacemeмt.
        STD     TVEC    Store as temporaray vector for player's character.
        LDD     VOUT    Get player's shot vector generated so far.
        SUBD    #C2     Give twice player's leftward displacement.
        STD     VOUT    Store as vector for player's shot.
        BRA     C73     Exit vector generating section.
U06     LDD     VOUT    If get here, slow speed is called for. Get shot vector peviously calculated.
        SUBD    #01     Give leftward displacement.
        STD     VOUT    Store as vector for player's shot.
        INC     RND2    Increment horizontal odd/even counter.
        LDA     RND2    Get odd/even counter.
        ANDA    #C1     If result here = 0, no movement this cycle.
        BEQ     C73     Branch out of vector generating section if result = 0.
        LDD     TVEC    Get player's vector generated so far.
        SUBD    #01     Give leftward displacement.
        STD     TVEC    Store as vector for player's character.
        BRA     C73     Exit vector generating section.
C72     CMPA    #45     If get here, no leftward movement called for . check for rightward movement. Value < 45 -no horiz movement.
        BLE     C73     No rightward movement called for. Branch and exit vector generating section.
        LDB     #C4     Shape number for player's character facing right.
        ADDB    PSHP    Add to shape already stored away.
        STB     PSHP    Store as proposed shape. Composite of 04 and old shape gives shape facing correct direction.
        CMPA    #57     Check for fast rightward motion.
        BLE     U07     Value less than 58, no fast movement. Branch to check for medium speed.
        LDD     TVEC    Fast rightward movement. Get player's vector already generated.
        ADDD    #C2     Add two pixel rightward displacement.
        STD     TVEC    Store as temporary vector for player's character.
        LDD     VOUT    Get player's shot vector generated so far.
        ADDD    #C4     Add twice player's displacement.
        STD     VOUT    Store as vector for player's shot.
        BRA     C73     Exit vector calculating section.
U07     CMPA    #51     Check for medium rightward speed. Value < 51 implies medium slow speed.
        BLT     U08     If less than 51, branch and do slow speed.
        LDD     TVEC    Get player's vector calculated previously.
        ADDD    #C1     Give slow rightward displacement.
        STD     TVEC    Store as temporary vector for player's character.
        LDD     VOUT    Get player's shot vector calculated before.
        ADDD    #02     Give twice player's rightward displacement.
        STD     VOUT    Store as vector for player's shot.
        BRA     C73     Exit vector calculating section.
U08     LDD     VOUT    Slow speed. Get player's shot vector previously calculated.
        ADDD    #C1     Add rightward displacement.
        STD     VOUT    Store as vector.
        INC     RND2    Increment horizontal odd/even counter.
        LDA     RND2    Get odd/even counter.
        ANDA    #01     If result from this = zero, no movement this cycle.
        BEQ     C73     Result = 0, branch.
        LDD     TVEC    Result not = 0. Get player's vector calculated so far.
        ADDD    #C1     Add rightward displacement.
        STD     TVEC    Store as temporary vector for player's character.
C73     LDD     TVEC    Get player's vector calculated above.
        BEQ     LL5     If the joystick was in the middle, the vector=0, player's character does not move. Branch.
        STD     12554   Vector not = 0. Store in c-list where player's vector is always stored (7,X).
```

```
        LDD     VOUT    Get player's shot vector calculated above. Can be sure vector not = 0 because player's vector not = 0.
        STD     SVEC    Store shot vector here. Note: if player's vector had been zero, shot vector wouldn't be modified.
        LDA     ,X      Get shape used for player in the last round.
        JSR     SHPADR  Look up the shape in the shape table.
        JSR     ANTISH  Erase the shape from the screen.
        JSR     NEWLOC  Compute a new proposed location from the player's vector and old screen location.
        JSR     REALCO  Translate the proposed location into real coordinates.
        LDA     PSHP    Get shape number determined for player when vector was calculated.
        JSR     SHPADR  Look up the shape's starting address.
        JSR     CKMOV   Check to see if the new shape will fit at the new screen location.
        BEQ     C28     If the new shape will fit, return code =  0, branch.
        JSR     RNDVEC  Will not fit at the new loc. Generate random vector. Try to make player bounce off of whatever is in the wa
        LDD     VOUT    Get the vector just generated.
        STD     12554   Store in c-list at location where player's vector is.
        JSR     NEWLOC  Generate new proposed screen loc from the new vector.
        JSR     REALCO  Translate into real coordinates.
        JSR     CKMOV   Check to see if the new location is unoccupied.
        BEQ     C28     If the new location is free, it is ok to move. Branch.
        LDD     12551   Still cant move. Give up. Get player's real screen location (4,X).
        STD     RLOC    Store as real screen loc so we can write the old shape back.
        LDA     12553   Get player's old bit set (6,X).
        STA     RBIT    Store as bit set so we can write the old shape back.
        LDA     12547   Get shape number for player's previous character.(,X)
        JSR     SHPADR  Look up the shapes address.
        JSR     WRTSHP  Write the old shape back. We have given up on the player this round. Cannot move him (or her).
        BRA     SHOT    Leave this section. Go check fire button.
C28     JSR     WRTSHP  If get here, was able to write new shape. Write the shape.
        LDA     PSHP    Get the number of the shape just written.
        STA     12547   Store in c-list as player's shape (,X).
        LDD     PSCR    Get the new screen location generated and written on.
        STD     12548   Store in c-list as player's screen location (1,X).
        LDD     RLOC    Get real value of location just written to.
        STD     12551   Store in c-list as player's real location (4,X).
        LDA     RBIT    Get bit set of location just written to.
        STA     12553   Store in c-list as player's bit set (6,X).
        BRA     SHOT    Information recorded with player's character. Branch to check fire button.
LL5     LDA     12547   If get here, joystick was in the middle. Shape not moved, but must rewrite in case a holemaker ate it.
        JSR     SHPADR  Look up player's old character shape address.
        LDD     12551   Get player's real screen loc so we can re-write the shape.
        STD     RLOC    Store as real location.
        LDA     12553   Get player's old bit set.
        STA     RBIT    Store as bit set.
        JSR     WRTSHP  Write player's old shape back.
*
* The joystick button will be checked to see if a shot is to be fired
*
SHOT    LDA     65280   Fire button is memory mapped. Read value.
        CMPA    #255    If value = 255, button not pushed.
        BEQ     CC1     Branch if not pushed.
        CMPA    #127    If value = 127, fire button not pushed.
        BEQ     CC1     Branched if not pushed.
        TST     RNDS    This is the button pushed last cycle flag. If it is true, button was already pushed.
        BNE     LLM     Branch if button already pushed.
        INC     RNDS    Set button pushed last cycle flag.
        LDD     SVEC    Get vector calculated for player's shot.
        STD     VOUT    Store as vector.
```

```
        LDD     12548   Get player's screen location. Shot must from originate from where player's character is.
        STD     PSCR    Store as proposed screen location. RBIT and RLOC still hold player's location.
        LDA     #48     Player's shot first shape number = 48. Shape number changes as shot proceeds.
        STA     PSHP    Store as proposed shape.
        LDA     #31     This is a counter for shot shape changes. Every eight cycles shape of player's shot will change.
        STA     TMP1    Store in space normally reserved for wobble byte.
        JSR     ADDCHQ  Add the player's shot to the character c-list.
        BRA     LLM     Branch around line below.
Q01     CLR     RND5    This line merely resets the button pushed last round flag. Only get here if button was not pushed.
*
* This next section checks to see if a holemaker can be added
* to the c-list. The decision is based on the number of mines
* already in the c-list and the number of holemakers allowed
* vs the number already in the c-list.
*
LLM     LDA     RND1    Get "random" number.
        ANDA    NUMM    Make the number <= than the number of mines on the screen.
        CMPA    #EEFH   Check against the number of mines neccesary before a holemaker can appear.
        BLT     TZ1     If the result is less, there will be no holemakers born this round. Branch.
        LDA     NUMH    Get the number of holemakers already living.
        CMPA    MAXH    Compare it with the maximum number allowed.
        BEQ     TZ1     If the maximum number of holemakers allowed already exit, branch.
        LDA     #102    Shape number for a holemaker.
        STA     PSHP    Store as proposed shape.
        LDD     12557   This is the screen location of the second character in the c-list.
        STD     PSCR    Store as proposed screen location. Chose second character's loc only because holemaker must start somewhere
        LDD     12560   Get second character's real memory location.
        STD     RLOC    Store as real location.
        LDA     12562   Get second character's bit set.
        STA     RBIT    Store as bit set.
        JSR     RNDVEC  Get a random vector.
        JSR     ADDCHQ  Add the holemaker to the character c-list.
        INC     NUMH    Increment the holemaker count for this round. Done adding holemaker.
*
* In this section a 'random' number is generated and checked
* against a parameter (MINCH) to see if a mine can be added
* to the c-list. No more than the maximum number allowed can
* be added.
*
TZ1     LDA     RND1    Adding mines to the c-list. Get random number.
        ORA     RND4    Or with a second random number.
        CMPA    MINCH   Compare with the chance of getting a mine this round.
        BHI     LV3     If the number is higher than MINCH, don't add a mine, branch.
        LDA     NUMM    Get number of mines presently alive on the screen.
        CMPA    MAXM    Compare with the maximum allowed this round.
        BEQ     CHARS   If we have reached the maximum number of mines allowed, don't add a mine, branch.
        LDA     #106    Character number for a mine.
        STA     PSHP    Store as proposed shape.
        LDD     MSCR    Get starting screen location for mines this round.
        STD     PSCR    Store as proposed screen location.
        LDD     MINR    Get real video ram location for mine calculated when setting up the round.
        STD     RLOC    Store as a real location.
        LDA     MINB    Get bit set for mine calculated when stting up this round.
        STA     RBIT    Store as bit set.
        LDA     #C1     Wobble byte for mines control whether the mine is actively chasing the player or bouncing (see mine section
        STA     TMP1    Store as proposed wobble byte.
```

```
        JSR     ADDCHC  Add the mine to the character c-list.
        INC     NUMM    Increment the number of mines counter.
        LDA     #$FF    Going to make a noise here to tell of the birth of a mine.
TZQ     DECA            Noise is a saw tooth wave of 255 decreasing periods.
        BEQ     CHARS   If A is = 0, done with noise, branch.
        TFR     A,B     Noise.
TZR     STB     $FF20   Store noise to A/D converter.
        DECB            Noise.
        BNE     TZR     More noise.
        BRA     TZQ     More noise.
*
* Values are tested here to see if the computer guns will fire.
*
LV3     CMPA    GCH     Here we see if guns fire. If mine was born, this was skipped. Compare contents of A with gun chance .
        BHI     CHARS   If A was higher, no guns are fired, branch.
        LDD     G1S     Get gun #1's screen location for this round.
        STD     PSCR    Store as screen location.
        LDD     G1R     Get gun #1's video ram loc as calculated when setting up this round.
        STD     RLOC    Store as real loc.
        LDA     G1B     Get gun 1#'s bit set as calculated when setting up this round.
        STA     RBIT    Store as bit set.
        LDD     G1V     Get gun #1's vector for this round.
        STD     VCUT    Store as vector.
        LDA     G1L     Get length of gun #1's shot for this round.
        STA     TMP1    Store as wobble byte. For guns, wobble byte is decremented to zero and then gun's shot deleted from c-list.
        LDA     #96     Shape number for computer gun shot.
        STA     PSHP    Store as proposed shape.
        JSR     ADDCHQ  Add shot to the character c-list.
        LDA     #136    Shape number for the gun itself. Going to write it on the screen whenever gun fires.
        JSR     SHPADR  Look up the shapes address.
        JSR     WRTSHP  Write the shape.
        LDD     G2S     Get gun #2's screen location.
        STD     PSCR    Store as screen location.
        LDD     G2R     Get gun #2's video ram loc as calculated in set up of round.
        STD     RLOC    Store as real location.
        LDA     G2B     Get gun #2's bit set as calculated in set up of round.
        STA     RBIT    Store as bit set.
        LDD     G2V     Get gun #2's vector.
        STD     VCUT    Store as vector.
        LDA     G2L     Get gun #2's shot length.
        STA     TMP1    Store as wobble byte.
        LDA     #96     Computer shot shape number.
        STA     PSHP    Store as proposed shape.
        JSR     ADDCHQ  Add computer shot to the c-list.
        LDA     #138    Shape number for gun #2.
        JSR     SHPADR  Look up shape address.
        JSR     WRTSHP  Write the gun.
*
* The loop below is entered once for every location in the c-list.
* Way up above the X register was set to 12547, the player's
* location in the c-list. The rest of the characters are now checked.
* By looping through nine bytes at a time each c-list entry is
* sampled. If the entry (shape number) is zero the slot is
* considered empty. Other numbers correspond to other characters:
*    ENTRY = CURC           ;Process current atom
*    ENTRY = #106           ;Process mine
```

```
*     ENTRY = #102          ;Process holemaker
*     ENTRY > #42           ;Process neutron
*     ALL ELSE              ;Process atom
* Each discriminating step occurs in the order given.
*
CHARS   LEAX    9,X     Add nine to X so we will be addressing the next character.
        CMPX    #13312  Check to see if the end of the c-list has been reached.
        LBGE    LOOP    If it has, branch way back up to LOOP.
        LDA     EWAI    Get the end of round wait counter.
        BEQ     N2Z     If it equals 0 (which it does till the last atoms are up) branch.
N3Z     DECA            Decrement wait value.
        BNE     N3Z     If not done waiting, branch back up.
N2Z     LDA     RND4    Gonna make the sound of the explosions if there has been one recently. RND4 is set to $FF decrmntd for soun
        BEQ     QC8     If RND4 = 0 , don't make any noise, branch.
        ANDA    #02     Going to make sound for two cycles every second cycle. Check to see if its time.
        BEQ     QCX     No, not this cycle, but branch to decrement counter anyway.
        LDA     RND1    Get that other random number.
        EORA    2,X     Do an XOR with 2nd byte of character's screen loc. Trying to make a psuedo random noise.
        ANDA    RND4    Add more confusion to the byte.
        COMA            Add more confusion to the byte.
QCX     DEC     RND4    Decrement RND4 so that the sound eventually dies out.
QC8     STA     $FF20   Store to the A/D converter. The zeros every two rounds are stored here too so the sound is dynamic.
        LDA     ,X      Done with the noise. Get the character number from the c-list.
        BEQ     CHARS   A character number of zero means no character, branch up.
        CMPA    CURC    Compare the character number with the current atom.
        BEQ     CHASED  If the character is CURC, branch to process CURC.
        CMPA    #106    Is the character a mine?
        LBEQ    MINE    If it is, branch to process a mine.
        CMPA    #102    Is the character a holemaker?
        LBEQ    HOLE    If it is, branch to process holemaker.
        CMPA    #42     All characters greater than or equal to 42 are shots of some sort at this point. Branch to do neutrons.
        LBGE    NEUT    Branch if neutrons.
        BRA     C30     The only characters not selected yet are bumbling, bothersome atoms. Branch to process them.
*
* Next section for current atom. Depending on random samples
* current atom (CURC) may run away from player's character
* (if shots have been fired) or head for location between
* computer guns (AIM) or possibly just continue with present
* vector.
*
CHASED  TST     RAWY    If get here, are processing current atom. Check run-away byte. (Incremented below by neutrons.)
        BLE     VC1     If not greater than zero, branch.
        DEC     RAWY    Decrement run away byte so it'll eventually get to zero and CURC can stop running.
        DEC     RAWY    Decrement again.
        LDA     RND1    Get the random number.
        ANDA    #07     If result from this is not zero, will skip running away this time and make CURC head for AIM location.
        BNE     VC1     Skip making CURC run away if not = 0.
        JSR     NEWVEC  Generate vector towards player.
        LDD     #0      Clean out D register. Going to make a vector away from player by subtracting vector from 0.
        SUBD    VOUT    Subtract generated vector.
        STD     7,X     Store as this CURC's vector.
        BRA     C30     Branch to process movement.
VC1     INC     RND1    In the section it is determined if CURC character we are processing will head for AIM.
        LDA     RND1    Number RND1 has been modified by line above. Get random number.
        ANDA    ATRCT   Bound number by the attraction parameter.
        BNE     C30     If the result was not zero, branch to process character's movement.
```

```
        JSR     CHNVEC  Generate vector towards AIM location.
        LDD     VOUT    Get vector generated.
        STD     7,X     Store as CURC character's vector.
*
* Both CURC and inactive atoms are processed the same in this
* section. Atom is moved according to vector if possible. Wobbling
* effect is taken care of.
*
C30     LDA     3,X     Here the movement of all atoms is processed. Get wobble byte for character from c-list.
        BEQ     C31     If wobble byte = 0 there is no wobble, don't bother processing the wobble, branch.
        LSLA            Multiply the wobble by two. This will be clear in a second.
        ANDA    #C2     Clean out all bits except the second. The result of this is added to the shape # to get # of wobble.
C31     ADDA    ,X      The antishape written will correspond to either the wobble or normal depending on which was written last.
        JSR     SHPADR  Look up the shape's address.
        JSR     ANTISH  Write the antishape to erase it from the screen.
        JSR     NEWLOC  Generate new screen loc from shapes old loc and vector.
        JSR     REALCO  Translate into real coordinates.
        LDA     3,X     Get wobble byte again. Going to check whether wobble or normal shape for this char is to be written next.
        BEQ     C32     If wobble byte = 0, next shape will be normal by default.
        DECA            Decrement the value for the wobble.
        LSLA            Multiply by two.
        ANDA    #C2     Remove all but second bit.
C32     ADDA    ,X      Add result to shape number to get wobble number if bit was set or keep shape number if not.
        JSR     SHPADR  Look up the shape's address.
        JSR     OKMOV   Check to see if the shape can be written at the location pointed to by RLOC and RBIT.
        BEQ     C33     If it can, branch.
        JSR     RNDVEC  If get here, shape wouldn't fit at new location. Generate random vector.
        LDD     VOUT    Get vector generated.
        STD     7,X     Store in c-list as character's vector.
        JSR     NEWLOC  Generate a new location from new vector and character's old postion.
        JSR     REALCO  Translate into real coordinates.
        JSR     OKMOV   Check to see if new location is clear for character to move into.
        BEQ     C35     If it is,branch.
        LDD     4,X     It wasn't, better to give up this cycle. Get shape's old real location.
        STD     RLOC    Store as real loc.
        LDA     6,X     Get shape's old bit set.
        STA     RBIT    Store as bit set.
        LDA     3,X     Get shape's wobble byte (note it was never modified).
        BEQ     C34     If wobble byte = 0 , then without a doubt character was in it's normal state before.
        LSLA            Wobble byte not = 0. Multiply by two.
        ANDA    #C2     Clear all but second bit.
C34     ADDA    ,X      Add character's normal state shape number.
        JSR     SHPADR  Look up resulting shape's address.
        JSR     WRTSHP  Write the old shape of the character back where we found it.
        LDA     ,X      We weren't able so move so apparently shape hit something. Reset wobble to shape wobbles for a while.
        ORA     3,X     Result of this op is assigned to the char's wobble byte. Note bit 1 is preserved, thus is old wobble state.
        STA     3,X     Store in c-list as character's wobble byte.
        LBRA    CHARS   All done with imoble atom, branch to process more characters.
C35     TST     3,X     We got here, we were able to move atom on the second try. Get character's wobble byte.
        BNE     C35A    If it was already wobbling, skip next instruction, else must set wobbling w/normal state being first.
        INC     3,X     Now the char's wobble byte = 1. Has to be odd so that after decrmnting below, correct antishp used next cyc
C35A    LDA     ,X      Get shape number. This'll be or'd with wobble byte and stored as wobble byte.
        ORA     3,X     Or with wobble byte. Note: can always be sure shape number is even.
        STA     3,X     Store result as wobble byte.
C33     TST     3,X     Get here, handle movement of all atoms whether they bounced or not. Check wobble byte.
        BEQ     C36     If wobble = 0 branch.
```

```
        DEC     3,X     Have done everything with a decremented value of wobble byte. Time to decrement the byte itself.
C36     JSR     WRTSHP  Finally! Write the shape (normal or wobble) determined for the atom.
        LDD     PSCR    Get the screen loc used.
        STD     1,X     Store in c-list as character's screen location.
        LDD     RLOC    Get real video ram location used.
        STD     4,X     Store in c-list as character's video ram location.
        LDA     RBIT    Get bit set used to write the shape.
        STA     6,X     Store in c-list as character's bit set.
        LBRA    CHARS   All done with atom, branch to get another character to work on.
*
* Neutrons and shots of all types are processed in the next section.
* If a shot hits something control is transferred to the BOMB section.
*
NEUT    STA     $FF20   If get here, character is either computer shot, player shot or chain reaction shot. Make a noise.
        STA     TMP3    Store the character (shape) number here. Will need it later as X register may be modified.
        CMPA    #96     Check to see if it's a computer shot.
        BEQ     NV1     If it is, branch around stuff below. For one, don't want computer shots causing atoms to run away.
        CMPA    #112    This is the number of character released when a mine explodes. It is temporarily inert.
        BEQ     COMW    If shot=112 branch below. It is explained there.
        INC     RAWY    Player's shots and chain reactions increment the RUN AWAY parameter.
NV1     JSR     SHPADR  For player's shots and chain reactions look up the old shape's address.
        JSR     ANTISH  Erase the shape.
        DEC     3,X     Dec wobble byte. If result=0, character is deleted.
        BNE     C701    Not = 0, branch.
        CLR     ,X      Delete character from c-list.
        LBRA    CHARS   Character deleted, go do another.
C701    LDA     ,X      The player's shot change every 7 cycles. Will see if dealing w/ player's shot, if so-is it time to change?
        CMPA    #48     Check against first shape # of player's shot.
        BLT     C70X    Less than, it must be a chain reaction neutron, shape = 42 branch.
        CMPA    #96     Check against computer's shot #.
        BEQ     C70X    Shot is computer's shot, branch.
        LDA     #C3     Have isolated shot to be one of the eight forms of the player's shot. 03 is a mask.
        ANDA    3,X     Every 4th cycle the result=0. When true, time to change player's shot character.
        BNE     C70X    Result not = 0, leave player's shape # the same this cycle.
        LDA     ,X      Result=0, time to change player's shot's shape. Get present shape #.
        ADDA    #06     Add 6 to get next shape #. Note: antibomb is at X+2 and bomb is at X+4.
        STA     ,X      Store as player's shot new shape #.
        JSR     SHPADR  Look up address of new shape.
C70X    JSR     NEWLOC  Character not deleted. Calculate new screen location.
        JSR     REALCO  Translate into real coorcinates.
        JSR     OKMOV   Check to see if it is clear to move into the new location.
        BNE     BOMB    If it isn't, blow up! Branch.
        JSR     WRTSHP  Write the shape at the new location.
        LDD     PSCR    Get screen loc used.
        STD     1,X     Store in c-list as character's screen loc.
        LDD     RLOC    Get real loc used.
        STD     4,X     Store in c-list as character's real loc.
        LDA     RBIT    Get bit set used.
        STA     6,X     Store in c-list as character's bit set.
        LBRA    CHARS   Branch to process new character.
*
* Exploding mines splinter into computer shots, however the
* shots are inert for a short time so that they may disperse
* from the point of the explosion. If this were not so, most
* of the shots would bump into one another and explode right
* there. The following section decides if it is time for the
```

```
* shots to become active.
*
COMW    DEC     3,X     When a mine explodes, a temp inert shape is used so shots can disperse w/out blowing up.
        BEQ     C2Q     If wobble byte just decremented=0, it is time to replace inert shots w/computer shots.
        JSR     SHPADR  It was not time, look up inert shots address in table.
        JSR     ANTISH  Erase the shape.
        JSR     NEWLOC  Calculate the new screen location from the old and the char's vector.
        JSR     REALCO  Translate into real coorcinates.
        JSR     CKMOV   Check to see if new loc is clear.
        LBNE    CHARS   If its not, don't bother restoring shape at old loc. Just branch.
        BRA     C7GX    It is ok, branch up and write the shape.
C2Q     LDA     #96     Get here, time to change inert shot into an active one so computer can score.
        STA     ,X      Store new shape # over old one at character's place in the c-list.
        LDA     #50     This 50 will be stored as wobble byte. Shot will last 50 cycles if it doesn't hit something.
        STA     3,X     Store as character's wobble byte.
        LBRA    CHARS   Go process another character.
*
* The next section takes care of the explosion which occurs when
* some kind of shot hits an object on the screen. The shape number
* for the shot is used to determine what kind of explosion
* to use. Say the shot or neutron's shape number is Q. The overlay
* shape number is then Q+2 and the explosion shape number is
* Q+4. The overlay shape is read from the screen, the explosion
* written and the overlay restored.
* This section also contains the code which decides if an atom
* has been fissioned or a mine destroyed.
*
BOMB    CLR     ,X      The shot hit something! Delete it from the character c-list.
        STX     TEMX    Going to need the X register, store it away temporarily.
        BRA     DCAR    Branch here in case DCAR section changed to subroutine. Worthless here though. Need I apologize?.
DRET    LDA     TMP3    Get the character number of the shot that caused the explosion.
        ADDA    #C4     Add 04 to get the shape number of the explosion.
        JSR     SHPADR  Get the address of the shape.
        JSR     WRTSHP  Write the shape.
        LDA     TMP3    Get the shot number again.
        ADDA    #C2     Add 02 to get the shape number for the overlay built in BOMYSH.
        JSR     SHPADR  Look up the address.
        JSR     WRTSHP  Write the overlay shape to restore the screen.
        LDA     #255    Gonna generate a 'pop'. This is for sound.
        STA     $FF20   Store to the D/A converter.
        STA     RND4    Store as random number. This is decerementec to make the explosion sound die away.
        LDA     #0      More noise.
        STA     $FF20   Store to the D/A converter
        LDA     #128    More noise.
        STA     $FF20   Store to the D/A converter.
        LDA     #255    More noise.
        STA     $FF20   Store to the D/A converter.
        LDX     TEMX    Get the value of the X register store previously.
        LBRA    CHARS   Branch to process another character.
DCAR    LDD     TLOC    We will now see if anyone is to be injured in the explosion. Get video loc where shot struck.
        SUBD    #13312  Start translating into a screen location. Subtract the start of video ram.
        LSLA            Multiply. There are four pixels per byte. We will multiply by 4 and add the bit set to get the screen loc.
        LSLB            Multply LSB.
        BCC     FF1     If no carry generated, skip next instruction. Doing 16 bit arithmetic.
        INCA            Propagate carry into MSB.
FF1     LSLA            Multiply by two again.
```

```
        LSLB            Multiply LSB.
        BCC     FF2     No carry then skip next instruction.
        INCA            Propagate carry into MSB.
FF2     ADDB    TBIT    Add bit set from point of collision into LSB. Now D contains the screen loc for where the shot struck.
        STD     PSCR    Store as screen loc.
        LDX     #12547  Load X with start of character c-list.
FF3     LDA     CURC    Load A with CURC's number for comparison below.
        LDB     #106    Load B with the character number for a mine.
FF4     LEAX    9,X     Make X point to the next character in the c-list.
        CMPX    #13312  Are we at the end of the c-list?
        LBGE    C52     If so, neither CURC nor MINE was involved in the explosion. Branch to finish up this section.
        CMPB    ,X      Check to see if character in c-list is a mine.
        BEQ     FF5     If it is, branch to see if it was close enough to the point of impact to be blown up.
        CMPA    ,X      Not a mine, check to see character in c-list is CURC.
        BEQ     FF5     If it is, check to see if it is close enough to the point of impact to be blown up. Branch.
        BRA     FF4     It is neither. A and B still hold values for CURC and MINE, branch part way up.
FF5     LDD     1,X     Are looking at either MINE or CURC in c-list. Get char's screen loc from the c-list.
        SUBD    PSCR    Subtract screen loc calculated above. Am going to do a cruce distance calculation.
        STB     RND4    Gonna get all bits = $FF80 and higher. Effectively the Y coordinate of the subtraction. Store LSB for later
        LSLA            Shift left. Want to get Y axis difference into one byte.
        LSLB            Shift the $80 bit off the left end.
        BCC     FF6     No carry - bit not set, branch around next instruction.
        INCA            Propage carry in LSB. Now have Y axis difference in the A register.
FF6     TSTA            Checking to see if difference is negative.
        BGT     FF7     If not, branch around next instruction.
        NEGA            Get absolute value.
FF7     CMPA    #04     Check to see if Y displacement is within 4 pixels. Note, this is a crude rectangular distance func.
        BGT     FF3     If Y displacement > 4 pixels, branch back up and continue checking the character c-list.
        LDA     RND4    Get here, Y within range. Gonna check X coordinate displacement.
        TFR     A,B     Make a working copy of X displacement.
        ANDA    #$7F    Zero out high bit.
        ANDB    #$4C    Have seven bit signed X cisplacement. Maximum distance=64 pixels. Check 7th bit to see if negative.
        BEQ     FF8     If not, skip next instructions.
        ORA     #$80    Propagate negative into 8th bit so byte is a proper negative.
        NEGA            Take two's complement to get X-axis difference.
FF8     CMPA    #C3     Check to see if X displacement is within 3 pixels of shot's point of impact.
        BGT     FF3     If not, branch up to check more characters in the c-list.
        LDA     ,X      The character was hit! Get character number from the c-list.
        CMPA    #106    Was it a mine?
        BNE     TZ9     If not, it was CURC. In that case branch below.
        JSR     SHPADR  Get mine's shape address.
        JSR     ANTISH  Erase mine from the screen.
        CLR     ,X      Delete the mine from the c-list.
        DEC     NUMM    Decrement the number of active mines counter.
        LDA     NUMN    Get the number of computer neutrons to be released from the explosion of the mine.
        STA     RND4    Store away. Will use as a loop counter.
        LDA     #112    Get inert computer shot #. Must make inert so that shots disperse without blowing up on each cthar.
        STA     PSHP    Store as proposed shape.
        LDD     1,X     Get deleted mine's screen location.
        STD     PSCR    Store as screen loc. Will make inert shots originate from where mine used to be.
        LDD     4,X     Get deleted mine's real location.
        STD     RLOC    Store as real loc.
        LDA     6,X     Get deleted mine's bit set.
        STA     RBIT    Store as bit set.
        LDA     #5      This will be stored as inert shot's wobble. In five cycles shot will become active computer shot.
        STA     TMP1    Store as wobble byte.
```

```
T2T     DEC     RND4    Decrement loop counter.
        LBEQ    C52     If equal to zero, we are done here. Branch below.
        JSR     RNDVEC  Get a vector for the inert shot.
        JSR     ADDCHC  Add the shot to the character c-list.
        BRA     T2T     Branch up to see if more are to be added.
T29     LDA     3,X     Get here, CURC is in explosion. Get char's wobble byte so can write correct antishape.
        BEQ     C702    If wobble byte = 0, then char is in it's normal state. (as opposed to wobble) Branch.
        LSLA            Shift wobble byte left to multiply by two.
        ANDA    #C2     Clear out all but second bit.
C702    ADDA    ,X      Add to character's shape # to get correct state.
        JSR     SHPADR  Look up shape in table.
        JSR     ANTISH  Erase the shape from the screen.
        CLR     ,X      Delete the character from the c-list.
        LDD     1,X     Get deleted char's screen loc. Going to make fission occur where character was.
        STD     PSCR    Store as screen loc.
        LDD     4,X     Get character's real loc from c-list.
        STD     RLOC    Store as real loc.
        LDA     6,X     Get character's bit set from c-list.
        STA     RBIT    Store as bit set.
        LDA     #42     First will add chain reaction neutrons to c-list if needed.
        STA     PSHP    Store character # for chain reaction neutrons to the c-list.
        STA     TMP1    Store as wobble byte too. This is decremented each cycle, so the neutron's max life is 42 cycles.
        LDA     CURC    Gonna check how many if any neutrons to add based on CURC's number. Lower nos. release none.
        CMPA    #22     Check to see if CURC is less than or equal to 22 (second atom).
        BLE     C46X    If it is, no neutrons added. Branch.
        JSR     RNDVEC  At least one to be added. Generate a vector.
        JSR     ADDCHC  Add the chain reactior neutron to the c-list.
        LDA     CURC    Gonna check CURC again.
        CMPA    #26     If CURC <= 26 (fourth atom) will not add any more neutrons.
        BLE     C46X    Less than or equal? Branch.
        JSR     RNDVEC  Generate vector for second neutron to be added.
        JSR     ADDCHC  Add character to c-list.
C46X    LDA     CURC    Get CURC. Gonna see if we reached the last atom for this round yet.
        ADDA    #04     Four + CURC gives number of next atom (if there is another).
        CMPA    #42     If the result is 42, we are on the last atom and no more are to be added this round.
        BNE     C46     If not the last, branch.
        LDA     EWAI    Just deleted an occurance of atom #6. There are no atoms added. Must make busy wait to slown things down.
        CMPA    #27     Just got end of round wait counter. If equals 27, i.e. 5 atoms left, will change attraction parameter.
        BNE     FUP     Not equal 25, branch around next instructions.
        LDA     #C2     New value of ATRACT. Now last five atoms will jump around less.
        STA     ATRCT   Store as attraction parameter.
FUP     INC     EWAI    Increment the end of round wait counter. Program will count to EWAI every loop to slow game as screen clear
        CLRA            Next character to be added is no character. Shape # 0.
C46     STA     PSHP    Rejoining instructions above. Store A as proposed shape.
        JSR     RNDVEC  Get a vector for the character.
        CLR     TMP1    Clear the wobble byte to be added to the c-list with the character.
        JSR     ADDCHC  Add new atom to c-list.
        JSR     RNDVEC  Must add one more. Get another vector.
        JSR     ADDCHC  Add another atom to the c-list.
        LDB     TMP3    Get shape number of shot that caused all this. Must give owner of the shot credit for the hit.
        CMPB    #42     Was it a chain reaction neutron?
        BNE     TA1     If not branch.
        LDA     CREAC   Get chain reactions this round counter.
        ADDA    #01     Add one to it for the hit.
        DAA             Put in proper BCD form.
        STA     CREAC   Return it where it came from.
```

```
        BRA     TA3     Branch below.
TA1     CMPB    #$6     Did a computer hit cause the explosion?
        BNE     TA2     If not, branch below. It would have to have been player's shot.
        LDA     CCMH    If it was computer shot, get computer hits this round counter.
        ADDA    #C1     Add one to give credit for the hit.
        DAA             Put in propoer BCD form (it is easier to print later).
        STA     CCMH    Store as computer hit counter.
        BRA     TA3     Branch below.
TA2     LDA     YURH    Get your hits counter.
        ADDA    #C1     Give yourself credit for the hit.
        DAA             Put in proper BCD format.
        STA     YURH    Put it back.
TA3     LDY     #12547  Must check to see if there is a new CURC after the fission.
        LDA     CURC    Get value of current atom.
C51     LEAY    9,Y     Step through character c-list.
        CMPY    #13312  Reach the end of the c-list.
        BGE     C250    If so, it is time to change CURC. Branch.
        CMPA    ,Y      Check CURC against character in c-list.
        BEQ     C52     If equal, there is at least one occurance of CURC left. Branch out of here, leave CURC alone.
        BRA     C51     Keep checking. Branch up to look at next character in the c-list.
C250    ADDA    #C4     If get here, it is time to change value of CURC. Add 4 to get number of next atom.
        CMPA    #42     If = 42 which is > 38 = atom #6, then this round is over.
        LBEQ    ZSTART  Branch to start a new round.
        STA     CURC    Store new value of CURC.
C52     LDA     TMP3    Rendevous in DCAR. Get # of shot that caused the explosion.
        ADDA    #C2     Add two to get number of overlay for the explosion.
        JSR     SHPADR  Look up the overlay shape's address.
        JSR     BOVYSH  Build the overlay at location of explsion.
        LBRA    DRET    Branch way up and finish this explosion mess up.
*
* Control is transferred to the following section whenever the player
* is struck by a mine.
*
DEAD    DEC     MENL    Player's character hit by mine. decrement numner of player's "men" left.
        LDA     #46     Gonna draw explosion shape on the screen.
        JSR     SHPADR  Look up shape's address in table.
        JSR     WRTSHP  Write the shape at the player's location.
        CLRA            Gonna make a whining noise.
LV1     DECA            A triangle wave of increasing amplitude and decreasing frequency for 255 cycles.
        BEQ     BR9     When A=0, sound is finished.
        TFR     A,B     Move A to B where it'll be incremented and sent to the D/A converter.
LV2     STB     $FF20   Store B to D/A converter.
        INCB            Increment like I said.
        BNE     LV2     Not = 0, still building this cycle of the wave.
        BRA     LV1     Get to here, this triangle built. Branch up.
BR9     LDA     #112    Done with noise. Gonna make pieces fly off the player's dead character.
        STA     TMP3    Go store the shape number of the first piece here temporarily.
        STA     RND4    Store the 112 here. Only reason is so can be sure RND4 does not equal 0. Explained below.
BR1     LDA     TMP3    Get the number of the piece added/not added to the c-list.
        ADDA    #C4     Add 4 to get the number of the next explosion piece to be added.
        CMPA    #136    If the piece to be added is 136, five pieces have already been added.
        BEQ     BR2     Exit this section if all five pieces have been added.
        STA     TMP3    Store the new shape number temporarily.
        STA     PSHP    Store it as proposed shape.
        TFR     A,B     Make a working copy. Are composing wobble byte- the byte says how many cycles the piece will live.
        ORB     #200    Make it live fairly long.
```

```
        STB     TMP1    Store as proposed wobble byte.
        JSR     RNDVEC  Generate a random vector.
        JSR     ADDCHG  Add the character to the c-list.
        BRA     BR1     Branch up to see if more pieces are to be added.
BR2     LDX     #12547  Gonna loop through the c-list and process only the pieces. Everthing else in the c-list remains intact.
BR3     LEAX    9,X     Make X point to the next character in the c-list.
        CLRB            Need a zero.
        STB     $FF20   Store it as sound.
        CMPX    #13312  Done looping through c-list?
        BNE     BR4     If not branch and loop some more.
        TST     RND4    Serving as a flag to tell when all pieces have been deleted. If = 0, all been deleted.
        BEQ     BR5     All deleted, branch below.
        CLR     RND4    Reset flag for all deleted. Will be set true if a piece is found in the c-list.
        BRA     BR2     Branch to start loop over.
BR4     LDA     ,X      Get the character # pointed to by X.
        BEQ     BR3     If zero, keep looping. No character there.
        CMPA    #116    Compare character number with the lowest of the pieces.
        BLO     BR3     If it's less, it's not one of the pieces. Leave it alone and branch up.
        ADDA    #02     It is one of the pieces. Add two to get the number of it's overlay shape.
        JSR     SHPADR  Find the shape's address.
        LDD     4,X     Get the piece's real loc.
        STD     RLOC    Store as real loc.
        LDA     6,X     Get the piece's bit set.
        STA     BIT     Store as bit set.
        JSR     WRTSHP  Write the overlay shape. (It was composed last cycle.)
        DEC     3,X     Decrement the piece's wobble byte.
        BNE     BR6     If not=0, shape gets to live at least one more cycle. Branch around next instructions.
        CLR     ,X      Delete the piece from the c-list.
        BRA     BR3     Branch to loop.
BR6     INC     RND4    Set the flag that says there are pieces still left in the c-list.
        LDA     #15     Gonna make a noise.
        ANDA    3,X     Value sent to A/D converter is piece's wobble byte's low 4 bits.
        STA     $FF20   Store to A/D converter.
        BNE     BR3     If result from last op = 0, will move the piece. Else leave alone. (this way pieces move slowly). Branch.
        JSR     NEWLOC  Gonna move piece. Find new location.
        JSR     REALCO  Translate to real coordinates.
        JSR     BOVYSH  Build an overlay of piece at new coordinates. note: shape's address found above.
        LDA     ,X      Get # of the piece. Have been working with the overlay till now.
        JSR     SHPADR  Find the shape's address.
        JSR     WRTSHP  Write the shape on the screen.
        LDD     RLOC    Get the generated real address.
        STD     4,X     Store in c-list as shape's real address.
        LDA     BIT     Get the bit set used.
        STA     6,X     Store in the c-list as the shape's bit set.
        LDD     PSCR    Get the screen loc generated.
        STD     1,X     Store in the c-list as the shape's screen address.
        LDA     3,X     Gonna make some noise. Get character's bit set.
        LDB     2,X     Get 2nd byte of character's screen loc.
        MUL             Scramble the two. (We have plenty of time at this point.)
        STA     $FF20   Store the MSB to the A/D converter.
        BRA     BR3     Branch up to loop more.
BR5     TST     MENL    Done with the pieces. See how many "men" the player has left.
        BNE     SSS     If there is at least one, the game isn't over yet. Branch below.
        JSR     TALLY   Player is done for. Put up the score.
SSS     JSR     CCISP   Whether its the end or not, put up the number of spares.
        JSR     KBDWAI  Wait for the player to read it and push the fire button.
```

```
        LBRA    RESTAR  Branch way way up.
*
* Following section processes a holemaker.
* If the holemaker hits something on the screen a jagged black
* hole is written where the collision occurred.
*
HOLE    JSR     SHPADR  If get here, are dealing with a holemaker. Acc A holds the number for a holemaker. Get the shape's address.
        JSR     ANTISH  Erase the holemaker from the screen so we can move it.
        JSR     NEWLOC  Generate the new screen location from the old and the holemaker's vector.
        JSR     REALCO  Translate into real coorcinates.
        JSR     OKMOV   Check to see if new loc is clear to move into.
        BEQ     LLA     If it is, branch below and make the move.
        JSR     RNDVEC  If it isn't, get a new vector for the holemaker.
        LDD     VOUT    Get the vector.
        STD     7,X     Store it in the c-list as the holemaker's vector.
        LDA     #104    This is the shape number for the hole. It will be drawn at the spot the holemaker struck something.
        JSR     SHPADR  Look up the hole's address.
        LDD     TLOC    Get the vidram loc where the holemaker struck something.
        STD     RLOC    Store as real loc.
        LDA     TBIT    Get the bit set where the holemaker struck something.
        STA     RBIT    Store as bit set.
        JSR     WRTSHP  Write the hole at this location. Note: the hole is like any other shape, just colored black.
        LBRA    CHARS   Branch up to process a new character.
LLA     JSR     WRTSHP  Get here, it was ok to move holemaker. Write the holemaker at the new location.
        LDD     RLOC    Get the real screen loc used.
        STD     4,X     Store in c-list as holemaker's real loc.
        LDD     PSCR    Get the screen location.
        STD     1,X     Store in the c-list as the holemaker's screen loc.
        LDA     RBIT    Get the holemaker's bit set used.
        STA     6,X     Store in the c-list as the holemaker's bit set.
        LBRA    CHARS   Branch to process another character.
*
* Section for processing mines.
* Mines continually plot a course for the player's character
* until they bump into something other than the player.
* The mines will then wander aimlessly for a number
* of cycles specified in the wobble byte (3,X). The number
* of cycles spent wandering is calculated between two parameters
* specified in the overlaid data.
*
MINE    JSR     SHPADR  Get here, dealing with a mine. Get the shape address.
        JSR     ANTISH  Erase the shape from the screen.
        TST     3,X     Check mine's wobble byte. If not=0, mine is wandering and not chasing the player.
        BNE     TZ3     If not = 0, branch to process a wandering mine.
        JSR     NEWVEC  Mine is chasing the player! Generate a vector towards the player's character.
        LDD     VOUT    Get the vector generated.
        STD     7,X     Store in the c-list as the mine's vector.
        JSR     NEWLOC  Generate the mine's new location based on the old location and the new vector.
        JSR     REALCO  Translate into real coordinates.
        JSR     OKMOV   Check to see if the new location is clear to move into.
        BEQ     TZ2     If it is, branch to make the move.
        LDD     TLOC    It's not, check to see if we bumped into the player.
        CMPD    12551   Check the real loc against the player's real loc in the c-list.
        BNE     TZ4     If didn't hit the player's character, branch.
        LDA     TBIT    May well have hit the player. Will check bit set to be sure.
        CMPA    12553   Compare bit set with player's bit set in the c-list.
```

```
        BNE     TZ4     Not the same, didn't hit the player dead on, branch.
        JSR     WRTSHP  Hit the player! Write the mine on the screen.
        JSR     DEAD    Player is dead! Go make sure he/she knows it.
TZ4     LDA     RAND4   Hit somthing and it wasn't the player's character. Mine must bumble around for a while.
        ECRA    RAND1   In the process of making up a new wobble byte for the mine.
        ANDA    MINSPL  Limit the number of cycles the mine will wander aimlessly by MINSPL.
        ORA     MINSPH  And make sure it wanders at least MINSPH cycles.
        STA     3,X     Store in the c-list as the mine's new wobble byte.
        JSR     RNDVEC  Generate a random vector for the mine.
        LDD     VOUT    Get the vector.
        STD     7,X     Store in c-list as the mine's new vector.
TZ3     DEC     3,X     Get here, processing all wandering mines.
        JSR     NEWLOC  Generate a new location from mine's vector and old location.
        JSR     REALCO  Translate into a real location.
        JSR     OKMOV   Check to see if new move is ok.
        BEQ     TZ2     If it is, branch.
        JSR     RNDVEC  No, try again. Generate another random vector.
        LDD     VOUT    Get generated vector.
        STD     7,X     Store in the c-list as the mine's vector.
        LDD     4,X     Get the mines real loc from the c-list. Gonna write the mine back where it was.
        STD     RLOC    Store as real loc.
        LDA     6,X     Store the mine's bit set.
        STA     RBIT    Store as bit set.
        JSR     WRTSHP  Write the mine back where it was found in the first place.
        LBRA    CHARS   Branch to process more characters.
TZ2     JSR     WRTSHP  New move was ok. Write the mine at the new location.
        LDD     RLOC    Get the real loc generated.
        STD     4,X     Store in the c-list as the mine's real loc.
        LDD     PSCR    Get the screen loc generated.
        STD     1,X     Store in the c-list as the mine's screen loc.
        LDA     RBIT    Get the bit set.
        STA     6,X     Store in the c-list as the mine's bit set.
        LBRA    CHARS   Branch to process more characters.
        END
```

```
        SETDP   $24     Direct page register loaded w/$24 in main program.
        CRG     $2FA2   Link by hand.
PSC2    EQU     $2447   LSB of screen location.
PSCR    EQU     $2446   Screen location.
RBIT    EQU     $2443   Bit set.
RLOC    EQU     $2444   Real video ram location.
STSH    EQU     $243E   Start of shape construction instructions.
TLOC    EQU     $2441   Working temporary for real vidram location.
TBIT    EQU     $2440   Working temporary for bit set.
VOU1    EQU     $243D   Second byte of vector.
VOUT    EQU     $243C   Vector.
PSHP    EQU     $243B   Shape number.
TMP1    EQU     $2432   Working storage.
CURC    EQU     $243A   Current fissionable character.
RAWY    EQU     $2439   Run away parameter.
TEMX    EQU     $2433   Working storage.
TVEC    EQU     $2437   Working vector temporary storage.
RND1    EQU     $242F   Working temporary.
STBO    EQU     $2402   * Start of screen layout border.
RND2    EQU     $242E   Working temporary.
RND3    EQU     $242D   Working temporary.
RND4    EQU     $242C   Working temporary.
RND5    EQU     $242B   Working temporary.
*
* ADDCHQ - Subroutine traverses character c-list looking for
* the first unfilled spot. When found, the following assignments
* are made:     PSHP -> ,X   ;character shape number
*               PSCR -> 1,X  ;screen location
*               TMP1 -> 3,X  ;wobble byte
*               RLOC -> 4,X  ;real vidram location
*               RBIT -> 6,X  ;bit set
*               VOUT -> 7,X  ;vector
* If there is no space in the c-list the character is not added
*
ADDCHQ  LDY     #12538  Start of character c-list - 9.
C90     LEAY    9,Y     Add nine to point to next character in c-list.
        CMPY    #13312  At the end of the c-list?
        BGE     C91     Yes, no character added, branch to exit.
        TST     ,Y      See if this space is occupied. If = 0, not occupied.
        BNE     C90     Occupied, branch up to loop more.
        LDA     PSHP    Get proposed shape number.
        STA     ,Y      Store in c-list as new shape.
        LDD     PSCR    Get proposed screen location.
        STD     1,Y     Store in c-list as new character's.
        LDA     TMP1    Get wobble byte.
        STA     3,Y     Store in c-list as character's.
        LDD     RLOC    Get real screen location.
        STD     4,Y     Store in the c-list as the character's.
        LDA     RBIT    Get bit set.
        STA     6,Y     Store in the c-list as the character's.
        LDD     VOUT    Get generated vector.
        STD     7,Y     Store in the c-list as the character's.
C91     RTS             Return.
*
* REALCO - Subroutine take screen loc from PSCR and translates
* into real coordinates. Vidram location is stored in RLOC
```

```
* and bit set is stored in RBIT.
*
REALCO  LDB     PSC2    Get 2nd byte of screen location.
        ANDB    #C3     Clear all bit last 2 bits - this makes up the bit set.
        STB     RBIT    Store as bit set.
        LDD     PSCR    Get screen loc again.
        LSRB            Divide by two. Gonna get vidram location.
        LSRA            Divide by two. Gonna get vidram location.
        BCC     CC1     If no carry generated, skip next instruction.
        ADDB    #$80    Propagate the carry into the low byte.
CC1     LSRB            Divide by two again.
        LSRA            Divide by two again.
        BCC     X12     If no carry generated, skip next instruction.
        ADDB    #$80    Propagate carry into the low byte.
X12     ADDD    #13312  Now have real loc assuming vidram is on page 0. Add 13312 to get page 34.
        STD     RLOC    Store as real loc.
        RTS             Return.
*
* OKMOV - Subroutine traces out shape onto the screen to see
* if the space is unoccupied. No pixels are written, only checked
* for coincidence. Input is present position of cursor indicated
* by RLOC and RBIT. STSH is accessed to get address of start of
* shape writing instructions. When and if it is determined that
* one of the pixels to be written would coincide with something
* on the screen already the condition code register is cleared
* and the subroutine returns. If there is no coincidence the
* zero flag is set before return.
*
OKMOV   LDD     RLOC    Checking to see if spot on screen is clear for a shape to move into. Get video ram location.
        STD     TLOC    Store in working temp.
        LDA     RBIT    Get bit set.
        STA     TBIT    Store in working temp.
        LDY     STSH    Load index register with first instruction for drawing the shape.
CC2     LDA     ,Y      Get shape construction instruction.
        BGE     CC3     If not less than zero branch around next instructions. An instruction less than zero means end of shape ins
        LDA     #$04    Get here, it is ok to move shape into RLOC with bit set RBIT. Gonna set zero flag in CC to say so.
        TFR     A,CC    Set zero flag.
        RTS             Return.
CC3     ANDA    #$40    Check instruction byte to see if pixel being written.
        BEQ     CC4     Result = 0, no pixel being written. Branch to bottom.
        LDA     #$C0    Gonna check pixel at TLOC with bit set TBIT. If there is already a lit pixel there- not OKMOV. $C0 is mask.
        LDB     TBIT    Get bit set. Will shift mask right TBIT times.
        BEQ     CC5     If TBIT = 0, no shifiting to be done, branch.
CC6     LSRA            Shift mask right.
        LSRA            Shift mask right.
        DECB            Decrement shift counter.
        BNE     CC6     If not = 0, not done shifting. Branch up.
CC5     ANDA    [TLOC]  And the mask with the vidram location.
        BEQ     CC4     If = 0, branch to move cursor and continue checking if the move is ok.
        CLRA            If the pixel wasn't blank then the move is not ok. Gonna set CC to say so.
        TFR     A,CC    Zero flag in condn code is not true - move was not ok.
        RTS             Return. Note: TLOC and TBIT contain location of coincidence. This is used for the origin of explosions.
CC4     JSR     NXTSET  Move 'cursor'.
        BRA     CC2     Branch up to check next instruction byte.
*
* NXTSET - Moves cursor pointed to by TLOC and TBIT, temporaries
```

```
* for RLOC and RBIT. The cursor bits of the shape instruction
* currently pointed to by the Y registor are consulted to find
* where next to move the cursor.
*
NXTSET  LDA     #$20    Subroutine for moving cursor. (see text for exact details.) Get bit 3 - if set cursor moves left.
        ANDA    ,Y      Check bit 3 of instruction byte.
        BEQ     CC7     Not set, branch below.
        DEC     TBIT    To move cursor left, decrement bit set.
        BGE     C10     If bit set greater than or = C, then TLOC is still correct. If not, (=-1) then must correct TBIT and TLOC.
        LDA     #C3     Bit set >0 and <4. If result above was -1 then new bit set = 3 and TLOC = TLOC - 1.
        STA     TBIT    Store as new temporary bit set.
        LDD     TLOC    Get vidram loc.
        SUBD    #C1     Subtract 1 to move one byte left.
        STD     TLOC    Put it back.
        BRA     C10     Moved left, assuming now that no need to check for moving to the right. Branch to check vertical movement.
CC7     LDA     #$1C    Bit 4 tells if cursor moves right. $10 is a mask.
        ANDA    ,Y      Mask instruction byte.
        BEQ     C10     If result=0, dont move cursor to the right Branch to check if cursor is to be moved up or down.
        INC     TBIT    Cursor moves right. Increment bit set.
        LDA     #04     Gonna compare with 4 (<=>0 and vidram loc+1).
        ANDA    TBIT    Equal to four?
        BEQ     C10     If not, branch to check vertical movement.
        CLR     TBIT    Get here, set bit set=0.
        LDD     TLOC    Gonna increment vidram loc.
        ADDD    #C1     Increment.
        STD     TLOC    Put it back.
C10     LDA     #$08    Mask to check for upward movement of cursor.
        ANDA    ,Y      Mask instruction byte.
        BEQ     C11     If result=0, branch to check for downward movement.
        LDD     TLOC    Get vidram loc.
        SUBD    #32     Move upward one line (32X4=128).
        CMPD    #13312  Is new loc below start of vidram?
        BGE     C14     If not, skip next instruction.
        ADDD    #3072   Cause wrap-around.
        BRA     C14     Branch to bottom.
C11     LDA     #$04    Mask for downward movemert.
        ANDA    ,Y      Mask out instruction byte.
        BEQ     CC9     If result=0, no downward movement, skip below.
        LDD     TLOC    Get vidram loc.
        ADDD    #32     Move down one line.
        CMPD    #16383  Did that move it off the end?
        BLE     C14     If not, branch around next instruction.
        SUBD    #3072   Cause wrap-around.
C14     STD     TLOC    Store vidram loc.
C09     LEAY    1,Y     Make Y point to next instruction byte.
        RTS             Return
*
* NEWLOC - Extracts character's vectore (7,X) and screen loc (1,X)
* and adds them together. Result is new screen loc stored in PSCR.
*
NEWLOC  LDD     1,X     Get character's screen loc.
        ADDD    7,X     Add character's vector.
        BGE     C15     If still > 0 (still on the screen maybe) branch.
        ADDD    #12288  Below zero not defined. Add 12288 (# of pixels on the screen) to cause wrap around.
        BRA     C16     Skip next instructions.
C15     CMPD    #12287  Off the end of the screen?
```

```
        BLE     C16     If not, branch to exit.
        SUBD    #12288  Cause wrap-around.
C16     STD     PSCR    Store new screen loc.
        RTS             Exit.
*
* ANTISH - Subroutine takes character's real vidram loc and bit
* set directly from the c-list and uses them as the starting
* cursor. The shape instructions pointed to by STSH are used
* to write the shape in black thereby erasing it from the screen
* completely.
*
ANTISH  LDD     4,X     Get character's vidram loc.
        STD     TLOC    Store in working temporary.
        LDA     6,X     Get character's bit set.
        STA     TBIT    Store in working temporary.
        LDY     STSH    Get the shape's address, put in index register.
C17     LDA     ,Y      Get instruction byte.
        BGE     C18     If not < 0, not end of shape.
        RTS             Byte < 0, end of shape. Return.
C18     ANDA    #$40    Check to see if pixel written.
        BEQ     C19     If not, branch below.
        LDA     #$C0    Yes. $C0 is a pixel mask.
        LDB     TBIT    Get the bit set.
        BEQ     C21     If = 0, no shifting of mask need be done.
C20     LSRA            Shift mask right.
        LSRA            Shift mask right.
        DECB            Decrement counter.
        BNE     C20     If not = 0, not done shifting. Branch up.
C21     COMA            Invert the mask.
        ANDA    [TLOC]  Anding causes all bits except masked bits to remain the same.
        STA     [TLOC]  Store it back into vidram. Pixel is now blacked out.
C19     JSR     NXTSET  Get next cursor location.
        BRA     C17     Branch up to do more.
*
* WRTSHP - Subroutine takes RLOC and RBIT as starting cursor.
* Shape who's instructions are pointed to by STSH is written
* on the screen.
*
WRTSHP  LDD     RLOC    Get vidram loc.
        STD     TLOC    Store in working temporary.
        LDA     RBIT    Get bit set.
        STA     TBIT    Store in working temporary.
        LDY     STSH    Put the shape's address in the index register.
C22     LDA     ,Y      Get instrution byte.
        BGE     C23     If not < 0, not done. Branch around next instruction.
        RTS             Return.
C23     TFR     A,B     Sent 2 copies of instruction byte.
        ANDA    #$40    Check to see if pixel being written.
        BEQ     C24     If not, branch to bottom.
        ANDB    #C3     Pixel being written. Isolate pixel in B.
        LDA     #C3     Shift left 3 minus bit set times to get pixel in correct location in byte.
        SUBA    TBIT    Make 3 minus bit set.
        STA     TVEC    Put here for counting.
        LDA     #C3     Pixel mask.
C300    TST     TVEC    See if need to shift.
        BEQ     C301    If = 0, no shifting to be done.
```

```
C302    LSLA            Shift mask left.
        LSLA            Shift mask left.
        LSLB            Shift pixel left.
        LSLB            Shift pixel left.
        DEC     TVEC    Decrement the counter.
        BNE     C302    Not done, branch up to do more.
C301    COMA            Done shifting. Invert mask.
        ANDA    [TLOC]  Mask out pixel (make black.)
        STA     [TLOC]  Put it back.
        ORB     [TLOC]  Add in shape's pixel.
        STB     [TLOC]  Put it back. Pixel is now written!
C24     JSR     NXTSET  Move cursor.
        BRA     C22     Branch up to do more.
VMULT4  LSL     VCUT    For vector multiplication. Not currently used.
        LSL     VCU1    Multiply vector LSB.
        BCC     V11     If carry clear skip next instruction.
        INC     VCUT    Propagate carry into MSB.
V11     RTS             Return.
        END
```

```
        SETDP   $24     Direct page register loaded w/$24 in main program.
        ORG     $2EB5   Linked by hand.
ADDCHQ  EQU     $2FA2   Sub for adding a character to the c-list.
ANTISH  EQU     $3088   Sub for erasing shape from screen.
NEWLCC  EQU     $3074   Sub for generating character's new screen loc from char's old loc and vector.
NXTSET  EQU     $3017   Sub for stepping moving 'cursor' while drawing a shape.
OKMOV   EQU     $2FE5   Sub for checking whether a proposed move puts char's shape in an unoccupied location on the screen.
REALCO  EQU     $2FC8   Sub for translating a screen loc into a real loc and bit set.
VMULT4  EQU     $30FA   Sub for multiplying a vector in VOUT.
WRTSHP  EQU     $30B8   Sub for writing a shape onto the screen.
PSC2    EQU     $2447   LSB of screen loctation.
PSCR    EQU     $2446   Screen location.
RBIT    EQU     $2443   Bit set of a real vidram location.
RLOC    EQU     $2444   Real video ram screen location.
STSH    EQU     $243E   Where shape's beginning addr is stored after a call to SHPADR.
TMP2    EQU     $2431   Working storage.
TLOC    EQU     $2441   Temp and working storage of RLOC.
TBIT    EQU     $2440   Temp and working storage of RBIT.
VOUT    EQU     $243D   LSB of vector.
VOUT    EQU     $243C   Vector output and storage.
TMP1    EQU     $2432   Working storage, usually used for wobblr byte.
TEMX    EQU     $2433   2 byte storage, usually for X register.
TVEC    EQU     $2437   Temporary vector storage.
SVEC    EQU     $2435   Shot vector storage.
RND1    EQU     $242F   Working storage.
TMP3    EQU     $2430   Working storage.
STBO    EQU     $2402   * Start of screen laycut for this round.
RND2    EQU     $242E   Working storage.
RND3    EQU     $242D   Working storage.
RND4    EQU     $242C   Working storage.
RND5    EQU     $242B   Working storage.
SHTBL   EQU     $241B   * Location of the shape table.
AIM     EQU     $241E   * Screen location where CURC is heading for, afinity based on ATRCT.
*
* SHPADR - The shape number is passed in the A register,
* the address of the shape is looked up in the shape table
* and the result is stored in STSH
*
SHPADR  TFR     A,B     Subroutine passed shape # through A, looks up shape's ador and stores in STSH.
        CLRA            Clean out high byte.
        ADDD    SHTBL   Add to the address of the shape table to get the address of the address of the shape.
        TFR     D,Y     Gonna do a sort of indirect address.
        LDD     ,Y      Get value at the offset in the shape table. This is the shape's address.
        STD     STSH    Store as start of shape.
        RTS             Return.
*
* DWNVEC - generates a vector towards the AIM location.
* Subroutine swaps player's screen position out of the c-list and
* replaces it with the AIM location and calls NEWVEC.
* NEWVEC thinks it is generating a vector towards the player.
* Upon return from NEWVEC the character's screen is restored
* to the c-list.
*
DWNVEC  LDD     12548   Generate a vectore towards AIM. Gonna store player's loc away for a minute.
        STD     TEMX    Store it away. Gonna call a sub that generates a vector towards player with a fake player loc.
        LDD     AIM     Get the point on the screen where CURC is heading.
```

```
        STD     12548   Store it as player's sceen loc for a minute.
        JSR     NEWVEC  Call sub that generates a vector towards the player.
        LDD     TEMX    Vector generated. Gonna put player's screen loc back where it was, no one will ever know we messed w/it.
        STD     12548   Store in player's screen loc in c-list.
        RTS             Go home.
*
* NEWVEC - subroutine gererates a vector towards the player's character
* from whatever character the X register is addressing in
* the c-list. Result is stored in VOUT.
*
NEWVEC  LDD     1,X     For generating a vector towards player. Get the character's screen loc from the c-list.
        ANDB    #$80    Clear all bits that describe the X coordinate of the character.
        STD     TVEC    Store that away temporarily.
        LDD     12548   Now get the player's screen loc from the c-list.
        ANDB    #$80    Clear all bits that describe the X coordinate.
        SUBD    TVEC    Subtract the player's Y coordinate from the character's.
        CMPD    #C      Are the on the same Y coordinate?
        BLE     C80     Same coord or player has higher Y coord branch around next instructions.
        LDD     #256    Making up vector. Give positive displacement of 2 pixels on Y-axis.
        BRA     C81     Branch to take care of X axis.
C80     BEQ     C81     Branch if same Y coord.
        LDD     #$FFC0  Player has lower Y coord than character. Vector will have -2 pixel Y axis displacement.
C81     STD     VOUT    Store the result of the Y coord comparison.
        LDB     2,X     Now do X coords. Get 2nd byte of character's screen location.
        ANDB    #$7F    Isolate bits that describe X coord.
        STB     TVEC    Store away for a nanosecond.
        LDB     12549   Get player's LSB of screen loc.
        ANDB    #$7F    Isolate bits that describe X coord.
        SUBB    TVEC    Subtract this from character's X coord.
        BNE     C82     If not the same X coord, branch.
        RTS             If the same, all done making up vector. Return.
C82     BGT     C83     If player is to the right, branch below.
        LDD     VOUT    Get vector generated so far.
        SUBD    #C2     Player is to the left. Subtract 2 to give 2 pxel leftward displacement.
        STD     VOUT    Store as resulting vector.
        RTS             Return.
C83     LDD     VOUT    Player is to the right. Get vector generated so far.
        ADDD    #02     Give 2 pixel rightward displacement.
        STD     VOUT    Store as resulting vector.
        RTS             Return.
*
* RNDVEC - Generates a 'random' vector. Result stored in VOUT.
*
RNDVEC  LDA     12554   Generating a 'random' vectorout of whatever is laying around.
        INC     RND1    Change this.
        ADDA    RND1    Modify that.
        ECRA    2,X     Scramble it up in the pan.
        STA     TVEC    And put it here for a minute.
        CLR     VOUT    Gonna make the vector in pieces. Want to start with a zero vector.
        CLR     VOU1    Clear second byte of vector.
        ANDA    #C1     Gonna build parts of the vector by checking the bits of the number we just made up.
        BEQ     VC1     If the 8th bit=0, branch.
        LDD     #$FF80  Not = 0, give -1 pixel Y displacement.
        STD     VOUT    Store Y displacement of vector.
        BRA     VC2     Go see about X displacement.
VO1     LDA     TVEC    Get the number generated above.
```

```
        ANDA    #C2     Check the 7th bit.
        BEQ     VC2     If it equals 0, the vector will have no vertical displacement at all. Branch.
        LDD     #128    Give vector +1 pixel Y displacement.
        STD     VOUT    Store as vector.
VC2     LDA     TVEC    Now X displacement.
        ANDA    #C4     Check 6th bit.
        BEQ     VC3     If = 0,no leftward X displacement. Branch.
        LDD     VOUT    Get vector generated so far.
        SUBD    #C1     Give leftward displacement.
        STD     VOUT    Store as vector.
        BRA     VC4     All done, branch to the bottom.
VC3     LDA     TVEC    Check for rightward displacement.
        ANDA    #C8     Check 5th bit.
        BEQ     VC4     If = 0, no X displacement. Branch.
        LDD     VOUT    Get vector generated so far.
        ADDD    #C1     Give rightward displacement.
        STD     VOUT    Store as vector.
VC4     LDD     VOUT    Dont want 0 vector, gonna check it. Get vector.
        BEQ     RNDVEC  If = 0, go up and start over.
        RTS             Non-zero vector generated. Return.
*
* BOVYSH - Builds an overlay shape from the pixels on the screen
* into the instructions pointed to by STSH. Building starts where
* the cursor is positioned, RLOC and RBIT. The shape instructions
* are actually mocified in the pixel bits.
*
BOVYSH  LDD     RLOC    Sub builds restore overlay for shape. Shape's addr in STSH, real loc in RLOC and bit set in RBIT.
        STD     TLOC    Store real vidram loc here for working storage.
        LDA     RBIT    Get bit set.
        STA     TBIT    Store as temporary bit set.
        LDY     STSH    Get the overlay shape's addr into the index reg.
YOO     LDA     ,Y      Get first instruction byte for overlay.
        BGE     YC1     If not less than zero, there is more to do. Branch.
        RTS             Instruction less than zero then done, return.
YO1     ANDA    #$40    Check to see if a pixel is written.
        BEQ     YC2     If result is zero, it is only a cursor movement instruction. Branch below.
        LDA     #$C0    This is a pixel mask.
        LDB     TBIT    Get the bit set. Will shift pixel mask right TBIT times.
        BEQ     YC3     If bit set = 0, skip next shift instructions.
YC4     LSRA            Shift right.
        LSRA            Shift right.
        DECB            Decrement bit set count.
        BNE     YC4     If B not equal to zero, more shifting to do. Branch up.
YC3     ANDA    [TLOC]  Mask at correct pixel. Get pixel from vidram.
        BEQ     YC7     If = 0, will merely clear pixel in instruction byte. Branch to do that.
        LDB     #C3     Pixel non-zero, gonna shift pixel value all the way to the right to build instruction byte.
        SUBB    TBIT    Subtract the bit set from the number 3.
        BEQ     YC5     If result = 0, pixel is all the way to the right ,branch.
YC6     LSRA            Shift right.
        LSRA            Shift right.
        DECB            Decrement the shift counter.
        BNE     YC6     If not done shifting, branch up.
YC5     LDB     ,Y      Get the instruction byte again.
        ANDB    #$FC    Clear pixel out of the byte.
        STB     ,Y      Put it back.
        ORA     ,Y      Or the pixel we just shifted into the instruction byte.
```

```
        STA     ,Y      Store result as new shape instruction.
        BRA     YC2     Branch down.
YC7     LDA     #$FC    Pixel was = 0 = black. load a mask into A.
        ANDA    ,Y      Get all but pixel from instruction byte.
        STA     ,Y      Store as instruction byte. Pixel = 0 = black.
YC2     JSR     NXTSET  Call subroutine to move 'cursor'.
        BRA     YC0     Branch up to do more.
        END
```

```
        SETDP   $24     Direct page register loaced w/$24 in main program.
START   EQU     $2500   The beginning of everything.
REALCO  EQU     $2FCB   Subroutine for translating a screen loc into a real vidram loc and bit set.
WRTSHP  EQU     $30B8   Subroutine for writing a shape on the screen.
SHPADR  EQU     $2EB5   Subroutine for fetching shape instruction address.
CREAC   EQU     $245A   Number of chain reactions.
COMH    EQU     $2459   Number of computer hits.
YURH    EQU     $2458   Number of your hits.
TCREAC  EQU     $2456   Total for chain reactions.
TCOMH   EQU     $2454   Total for computer.
TYURH   EQU     $2452   Total for player.
MENL    EQU     $2451   Player's men left.
PSCR    EQU     $2446   Proposed screen location.
TMP2    EQU     $2431   Working temporary.
TMP1    EQU     $2432   Working temporary.
TEMX    EQU     $2433   Working temporary.
RND1    EQU     $242F   Working temporary.
TMP3    EQU     $2430   Working temporary.
RND2    EQU     $242E   Working temporary.
RND3    EQU     $2420   Working temporary.
RND4    EQU     $242C   Working temporary.
RND5    EQU     $2428   Working temporary.
ME1     EQU     $2458   Address of "Spares:".
ME2     EQU     $245D   Address of "game over".
ME3     EQU     $245F   Address of "This Round:"
ME4     EQU     $2461   Address of "Chain Reactons".
ME5     EQU     $2463   Address of "Computer Hits"
ME6     EQU     $2465   Address of "Your Hits".
ME7     EQU     $2467   Address of "Reaction Total".
ME8     EQU     $2469   Address of "Computer Total".
ME9     EQU     $246B   Address of "Your Total".
ME10    EQU     $246D   Address of "Your High Score".
ME11    EQU     $246F   Address of "00".
MEXX    EQU     $2471   Address where text string for numbers to be displayed is built.
YURHS   EQU     $2473   Your high score stored here.
TYURHL  EQU     $2453   Low byte of your score.
TCOMHL  EQU     $2455   Low byte of computer score.
TREACL  EQU     $2457   Low byte of chain reaction score.
SETPTR  EQU     $2400   * Address of next set of overlaid dta.
        ORG     $2CC4   Start here.
*
* TALLY - Subroutine puts up the scoreboard and the updated
* scores. Also writes 'game over' when applicable.
*
TALLY   LDD     #1537   For putting up score board. 1537 is screen loc where the message "This Round:" will go.
        STD     PSCR    Store as screen loc.
        LDX     ME3     Get the start of the message into the index pointer.
        JSR     WRTMES  Write it on the screen.
        LDD     #2B17   Screen loc where "Chain Reactions" will go.
        STD     PSCR    Store as screen loc.
        LDX     ME4     Get the start of the message into the index pointer.
        JSR     WRTMES  Write it on the screen.
        LDD     #4097   Screen loc where "Computer Hits" will go.
        STD     PSCR    Store as screen loc.
        LDX     ME5     Get the start of the message into the index pointer.
        JSR     WRTMES  Write it on the screen.
```

```
LDD     #5377   Screen loc where "Your Hits" will appear.
STD     PSCR    Store as screen loc.
LDX     ME6     Get the start of the message into the index pointer.
JSR     WRTMES  Write it on the screen.
LDD     #7041   Screen loc where "Reaction Total" will go.
STD     PSCR    Store as screen loc.
LDX     ME7     Get the start of the message into the index pointer.
JSR     WRTMES  Write it on the screen.
LDD     #8321   Screen loc where "Computer Total" will go.
STD     PSCR    Store as screen loc.
LDX     ME8     Get the start of the message into the index pointer.
JSR     WRTMES  Write it on the screen.
LDD     #9601   Screen loc where "Your Total" will go.
STD     PSCR    Store as screen loc.
LDX     ME9     Get the start of the message into the index pointer.
JSR     WRTMES  Write it on the screen.
LDD     #10881  Screen loc where "Your High Score" will appear.
STD     PSCR    Store as screen loc.
LDX     ME10    Get the start of the message into the index pointer.
JSR     WRTMES  Write it on the screen.
LDD     #7148   Screen loc where "00" will go.
STD     PSCR    Store as screen loc.
LDX     ME11    Get the start of the message into the index pointer.
JSR     WRTMES  Write it on the screen.
LDD     #8428   Screen loc where "00" will go.
STD     PSCR    Store as screen loc.
LDX     ME11    Get the start of the message into the index pointer.
JSR     WRTMES  Write it on the screen.
LDD     #9708   Another "00".
STD     PSCR    Store as screen loc.
LDX     ME11    Get the start of the message into the index pointer.
JSR     WRTMES  Write it on the screen.
LDD     #10988  Screen loc where another "00" will go.
STD     PSCR    Store as screen loc.
LDX     ME11    Get the start of the message into the index pointer.
JSR     WRTMES  Write it on the screen.
CLRA            Clear out high byte of D register.
LDB     CREAC   Get # of chain reactions. Gonna build a text string.
JSR     BCBUFF  Build string for WRTMES from D register.
LDD     #2919   Where # of chain reactions will go.
STD     PSCR    Store as screen loc.
LDX     MEXX    Get start address of number into X register.
JSR     WRTMES  Write the number on the screen.
CLRA            Clear out high byte of D register.
LDB     CCMH    Get # of computer hits.
JSR     BCBUFF  Turn it into a text string for WRTMES.
LDD     #4199   Where # of computer hits will go.
STD     PSCR    Store as screen loc.
LDX     MEXX    Get start address of number into the X register.
JSR     WRTMES  Write the number on the screen.
CLRA            Clear out high byte of D register.
LDB     YLRH    Get # of your hits this round.
JSR     BCBUFF  Turn it into a text string for WRTMES.
LDD     #5479   Screen loc where # of player's hits will appear on the screen.
STD     PSCR    Store as screen loc.
LDX     MEXX    Get start address of number into the X register.
```

```
        JSR     WRTMES  Write the number on the screen.
        LDB     TCREAC  Load bytes of chain rection total into D in reverse order to do BCD arithmetic.
        LDA     TREACL  Load low byte into high byte.
        ADDA    CREAC   Add chain reactions this round to low byte of chain reaction total.
        JSR     BFIX    Go do DAAs if needed.
        ADDA    CREAC   Gonna add chain reaction total again because they score three times per hit.
        JSR     BFIX    Take care of BCD arithmetic.
        ADDA    CREAC   Add for the third and last time.
        JSR     BFIX    Adjust for third and last time.
        EXG     A,B     Done adding to reaction total. Put bytes back in order.
        STD     TCREAC  Store it away.
        JSR     BCBUFF  Turn it into a text string for WRTMES.
        LDD     #7132   Screen loc where chain reaction total goes.
        STD     PSCR    Store as screen loc.
        LDX     MEXX    Get start address of number into the X register.
        JSR     WRTMES  Write the number on the screen.
        LDA     #C6     Going to score computer hits. Computer gets six times the credit per hit.
        STA     TMP3    Put the six here and use as a counter.
        LDB     TCOMH   Gonna add computer hits the way we added chain reactions except gonna do it 3 times instead of 2.
        LDA     TCOMHL  Put low byte into high byte.
UFF     DEC     TMP3    Count down to zero.
        BEQ     UGG     If equal to zero, done adding computer hits. Branch.
        ADDA    CCMH    Add computer hits this last round.
        JSR     BFIX    Adjust decimals.
        BRA     UFF     Branch up to do more.
UGG     EXG     A,B     Put the D register in the correct order.
        STD     TCOMH   Store for posterity.
        JSR     BCBUFF  Turn it into a text string for WRTMES.
        LDD     #8412   Screen loc where Computer Total goes.
        STD     PSCR    Store as screen loc.
        LDX     MEXX    Get start address of number into the X register.
        JSR     WRTMES  Write the number on the screen.
        LDB     TYURH   Player only gets 2 times number of hits made.
        LDA     TYURHL  Put low byte into high byte.
        ADDA    YLRH    Add hits this round.
        JSR     BFIX    Take care of BCD arithmetic.
        ADDA    YLRH    Add one more and a final time.
        JSR     BFIX    Put the totals back into BCD form.
        EXG     A,B     Put bytes back into correct order.
        STD     TYURH   Put player's score away.
        JSR     BCBUFF  Turn it into a text string for WRTMES.
        LDD     #9692   Screen loc where player's score appears.
        STD     PSCR    Store as screen loc.
        LDX     MEXX    Get start address of number into the X register.
        JSR     WRTMES  Write the number on the screen.
        TST     MENL    Check and see if player has any men (or women) left.
        BNE     LC4     If so, branch down.
        LDD     TYURH   No, gonna see if player's score is player's high score.
        CMPD    YLRHS   Compare to the old high score.
        BLO     LC4     If less branch around.
        STD     YLRHS   New high score! Store it as high score.
LC4     LDD     YLRHS   Gonna write high score on the screen.
        JSR     BCBUFF  Turn it into a text string for WRTMES.
        LDD     #10972  Screen loc where high score goes.
        STD     PSCR    Store as screen loc.
        LDX     MEXX    Get start address of number into the X register.
```

```
        JSR     WRTMES  Write the number on the screen.
        TST     MENL    Check player's men left.
        BEQ     L05     If there are none left, branch around next instructions.
        LDD     SETPTR  Time to see if player gets a free ball. Happens every 2nd round, dependent on SETPTRs low byte.
        ANDB    #C1     Because SETPTR is odd, free ball every 2nd round.
        BEQ     XQU     If equal to zero, no free ball this round. Branch around next instructions.
        INC     MENL    Give player a free ball.
XQU     JSR     CDISP   Put up number of spares .
        CLR     CREAC   Zero out chain reaction count for next round.
        CLR     CCMH    Zero out computer hit count for next round.
        CLR     YURH    Zero out your hit count for next round.
        JSR     KBDWAI  Subroutine waits for the fire button.
        RTS             Go home.
L05     LDD     #41     Screen loc where "game over" goes.
        STD     PSCR    Store as screen loc.
        LDX     ME2     Get address of "game over" into X register.
        JSR     WRTMES  Write it on the screen.
        JSR     KBDWAI  Wait for the fire button.
        PULS    X       At present, we are in a subroutine. This will pull two bytes off the hardware stack and <more>
        LBRA    START   simulate an RTS without changing the program counter. Now we can branch safely without overflowing stack.
*
* WRTMES - puts a string of characters up on the screen (always
* text in this application). Leading zeros are stripped out of
* the text. The start of the string is pointed to by the X register,
* the end denoted by a shape number of zero. Each shape in the
* string is placed on the screen 5 spaces apart.
*
WRTMES  CLR     RND4    Sub for writing messages. RND4 is a flag to wipe out leading zeros. (see below)
MC0     LDA     ,X+     Get byte of message.
        BGE     MC1     In this case if byte>=0, done with message.
        CMPA    #140    Is it a "0" (text zero that is).
        BEQ     M10     If so, branch so as not to set RND4.
        INC     RND4    Make RND4 non-zero.
M10     TST     RND4    If flag is set, a character not = 0 has been encountered.
        BNE     M11     Not equal to zero, skip next instruction.
        LDA     #208    A leading zero! substitute for a blank. (blank = 208)
M11     JSR     SHPADR  Look up the character shape's address.
        JSR     REALCO  Translate the value in PSCR into bit set and vidram location.
        JSR     WRTSHP  Write the character.
        LDD     PSCR    Gonna move to the right, Get the screen loc.
        ADDD    #05     Move to the right 5 pixels.
        STD     PSCR    Put it back for the next character.
        BRA     MC0     Branch up to do more.
MC1     RTS             Done. Return.
*
* BCBUFF - A bcd number is passed in the D register and turned
* into the corresponding text string for display by WRTMES.
* The resulting 4 byte string is written at MEXX.
*
BCBUFF  STD     TEMX    For turning a BCD number into text. D reg holds the number. Store it here.
        LDX     MEXX    Get address of location where will build the string.
        ANDA    #$F0    Isolate first BCD digit of number.
        LSRA            Shift right. Must get digit so we can add to 140 (="0") and end up with a shape number.
        LSRA            Shift right again. Note: shape number must be even.
        LSRA            Shift again.
        ADDA    #140    Add to number for "0" to offset.
```

```
        STA     ,X+     Store in area where string is being built.
        LDA     TEMX    Get the original BCD number again.
        ANDA    #$0F    Get 2nd digit.
        LSLA            Shift left to get an even number.
        ADDA    #140    Add offset of "0". Note: 140="0", 142="1", 144="2" ...
        STA     ,X+     Store in area where string is being built.
        LDD     TEMX    Get original BCD number again.
        TFR     B,A     Make a copy of the low byte ...
        STA     TEMX    ... and put it here where we can get it easily.
        ANDA    #$F0    Isolate 3rd BCD digit.
        LSRA            Move right as with first digit.
        LSRA            Shift right.
        LSRA            Shift right.
        ADDA    #140    Add offset of character "0".
        STA     ,X+     Store where string is being built.
        LDA     TEMX    Get byte stored away a nano-second ago.
        ANDA    #$0F    Isolate 4th (last) digit.
        LSLA            Shift left to get an even number.
        ADDA    #140    Add the offset.
        STA     ,X+     Put it where string is being built (last character at MEXX)
        CLRA            Need a zero.
        STA     ,X      Store in string to signal end of string.
        RTS             Go home.
*
* KBDWAI - Goes into a busy wait and returns when the fire
* button in pushed.
*
KBDWAI  LDA     65280   Get location where fire button is mapped.
        CMPA    #255    If = 255, button not being pushed.
        BEQ     KBDWAI  Not being pushed then branch up.
        CMPA    #127    If = 127, button not being pushed.
        BEQ     KBDWAI  Not being pushed then branch up.
        RTS             Button was pushed. Return.
*
* BFIX - performs the DAA instruction on the whole D register
* (it is only defined for A). Subroutine is called with high byte
* and low byte of D already in reverse order.
*
BFIX    DAA             Routine for doing 16 bit BCD arithmetic. Decimal adjust whatever is in A.
        BCC     TU1     If no carry, branch to bottom to return.
        EXG     A,B     Switch A and B so can use DAA instruction on the lower byte.
        ADDA    #C1     Add carry into byte.
        DAA             Decimal adjust.
        EXG     A,B     Put the bytes back into reversed order.
TU1     RTS             Return.
*
* CDISP - If the player has any spares left this routine
* is called to display them in the upper left hand corner
* of the screen.
*
CDISP   LDD     #129    Where message "Spares:" will go.
        STD     PSCR    Store as screen loc.
        LDX     ME1     Address where message "Spares:" starts.
        JSR     WRTMES  Write the message.
        LDD     #550    Location where the shape for player's character will be written to display spares.
        STD     PSCR    Store as screen loc.
```