```
        .bss    ; the below calculated by init()
Lstatic:
        .space  2 ;2            ;uint16_t Lstatic[20];
        .space  2 ; 24
        .space  2 ;    0
        .space  2 ;      6+1
Ltblsiz:
        .space  2
Lgoalpwr:
        .space  2
        .space  2 ;8
        .space  2 ;  6
Lincrem:
        .space  2 ;    0        ;uint16_t* Lincrem = &Lstatic[8];
Lstrtvl:
        .space  2 ;     24+1    ;uint16_t* Lstrtvl = &Lstatic[9];
Lstopvl:
        .space  2 ;8            ;uint16_t* Lstopvl = &Lstatic[10];
Lfourth:
        .space  2 ;  0          ;uint16_t* Lfourth = &Lstatic[11];
        .space  2 ;    6
        .space  2 ;     30+1
Lioptrs:
Lgetptr:
        .space  2
Lputptr:
        .space  2
        .space  2 ;2
        .space  2 ;  0
        .space  2 ;   24
        .space  2 ;     30+1


        .equ    MAXROWS,4       ;const uint16_t MAXROWS = 4;
        .equ    COLS,4          ;const uint16_t COLS = 4;
Lboard:
        .space  2*MAXROWS*COLS  ;uint16_t Lboard[MAXROWS*COLS];

        .text
        .global init
        ;; upon return: Lstatic
        ;; area (except Lioptrs)
        ;; initialized for table
        ;; with the indicated
        ;; # rows and goal tile
init:
        mov     W1,Lgoalpwr     ;void init(uint16_t rows /*W0*/,
        mov     #Lincrem,W1     ;          uint16_t goal /*W1*/) {
        mov     #2*COLS,W2      ; uint16_t* i /*W1*/ = Lincrem;;
        mov     #0x0002,W3      ; Lgoalpwr = goal;
        mov     W3,[W1-4*COLS]  ; i[-2*COLS] = 2; // down key pressed
        mov     W2,[W1-4]       ; i[-2] = 2*COLS; // right
        mov     W2,[W1+4]       ; i[2] = 2*COLS; // left
        mov     W3,[W1+4*COLS]  ; i[2*COLS] = 2; // up

        mov     #(COLS-1)*2,W2  ; // W2 now (COLS-1)*2 e.g. 6
        dec     W0,W4           ;
        sl      W4,#3,W4        ; // W4 now COLS*(rows-1)*2 e.g. 24
        clr     W5              ; // W5 now 0

        mov     #Lstrtvl,W1     ; i = Lstrtvl;
        mov     W4,[W1-4*COLS]  ; i[-2*COLS] = COLS*(rows-1)*2; // down key
        mov     W2,[W1-4]       ; i[-2] = (rows-1)*2; // right
        mov     W5,[W1+4]       ; i[2] = 0; // left
        mov     W5,[W1+4*COLS]  ; i[2*COLS] = 0; // up

        mov     #Lstopvl,W1     ; i = Lstopvl;
        mov     W5,[W1-4*COLS]  ; i[-2*COLS] = 0; // down key pressed
        mov     W5,[W1-4]       ; i[-2] = 0; // right
        mov     W2,[W1+4]       ; i[2] = (rows-1)*2; // left
        mov     W4,[W1+4*COLS]  ; i[2*COLS] = COLS*(rows-1)*2; // up

        add     W4,#2*COLS,W0   ; // W0 now COLS*(rows-1)*2+COLS*2=COLS*rows*2
        mov     #2*COLS-1,W2    ; // W2 now 2*COLS-1 e.g. 7
        dec     W0,W3           ; // W3 now COLS*rows*2-1 e.g. 31
        inc     W4,W4           ; // W4 now COLS*(rows-1)*2+1 e.g. 25

        mov     #Lfourth,W1     ; i = Lfourth;
        mov     W2,[W1-4*COLS]  ; i[-2*COLS] = 2*COLS-1; // down key pressed
        mov     W4,[W1-4]       ; i[-2] = COLS*(rows-1)+1; // right
        mov     W3,[W1+4]       ; i[2] = COLS*rows*2-1; // left
        mov     W3,[W1+4*COLS]  ; i[2*COLS] = COLS*rows*2-1; // up

        mov     W0,Ltblsiz      ; Ltblsiz = COLS*rows*2;
        return                  ;} // init()

empties:
        ;; upon return: W0 >= 0
        ;; holds count of zero-
        ;; valued cells in table
        add     W0,W1,W1        ;uint4_t empties(uint16_t* tbl /*W0->W2*/,
        mov     W0,W2           ;                uint16_t size /*W1*/) {
        clr     W0              ; uint16_t retval /*W0*/ = 0, * curr /*W1*/;
Lnext0:
        cp0     [--W1]          ; for (curr = &tbl[size]; curr > &tbl[0]; )
        bra     NZ,Lnot0        ;  if (*--curr == 0)
        inc     W0,W0           ;   retval++;
Lnot0:
        cp      W1,W2           ;
        bra     GT,Lnext0       ;
        return                  ;} // empties()

gamewon:
        ;; upon return: 0<=W0<=1
        ;; holds logic value
        ;; (any cell >= goal)
        add     W0,W1,W1        ;uint1_t gamewon(uint16_t* tbl /*W0->W3*/,
        mov     W0,W3           ;                uint16_t size /*W1*/,
        mov     #1,W0           ;                uint16_t goal /*W2*/) {
Lnextc:
        cp      W2,[--W1]       ; uint16_t* curr /*W1*/;
        bra     LE,Ldidit       ; for (curr = &tbl[size]; curr > &tbl[0]; )
        cp      W1,W3           ;  if (*--curr <= goal)
        bra     GT,Lnextc       ;   return 1;
        clr     W0              ; return 0;
Ldidit:
        return                  ;} // gamewon()

apply:
        ;; upon return: W0 holds
        ;; points earned by tilt
        ;; of the table toward
        ;; angle (<-2|-2|2|>2)
        ;; indicated by W1 using
        ;; the lookup values
        ;; stored in Lstatic to
        ;; avoid repeat calcs.
        push.d  W8              ;uint16_t apply(uint16_t* tbl /*W0->W8*/,
        push.d  W10             ;               int16_t delta /*W1:d|r|l|u*/
        mov     W0,W8           ;{
        clr     W0              ; uint16_t points /*W0*/ = 0;
        mov     #Lincrem,W6     ;
        sl      W1,W7           ;
        mov     [W6+W7],W2      ; uint16_t* incr /*W2*/, * start /*W3*/,
        mov     #Lstrtvl,W6     ;         * stop /*W4*/, * fourth /*W5*/;
        mov     [W6+W7],W3      ;
        mov     #Lstopvl,W6     ; incr = *((uint16_t*) Lincrem + delta);
        mov     [W6+W7],W4      ; start = *((uint16_t*) Lstrtvl + delta);
```

```
        mov     #Lfourth,W6     ; stop = *((uint16_t*) Lstopvl + delta);
        mov     [W6+W7],W5      ; fourth = *((uint16_t*) Lfourth + delta);
L2pass:
        add     W8,W3,W6        ; do { // for every row (or column) in the table
        add     W8,W4,W11       ;  for (uint16_t* cell /*W6*/ = &tbl[start];
Lscoot:
        cp      W6,W11          ;   cell != &tbl[stop] /*W11*/; cell += delta) {
        bra     Z,Lcollap       ;   uint16_t non0 /*W7*/;
Lslid0:
        cp0     [W6]            ;   do {
        bra     NZ,Lnblank      ;    if (*cell == 0) {
        clr     W7              ;     non0 = 0;
        mov     W6,W10          ;     for (uint16_* slider /*W10*/ = cell;
Loverw0:
        mov     [W10+W1],[W10]  ;         slider != &tbl[stop]; cell += delta) {
        ior     W7,[W10],W7     ;      *slider = *(slider + delta);
        add     W10,W1,W10      ;       non0 |= *slider;
        cp      W10,W11         ;
        bra     NZ,Loverw0      ;     }
        clr     [W11]           ;     *slider = 0;
        bset    W0,#0           ;     points |= 1; // credit for scoreless scoot
        cp0     W7              ;    } else break; // *cell != 0
        bra     NZ,Lslid0       ;   } while (non0);
Lnblank:
        add     W6,W1,W6        ;  }
        bra     Lscoot          ;  // first pass complete: tilted to separate 0s
Lcollap:
        add     W8,W3,W6        ;  for (uint16_t* cell/*W6*/= &tbl[start];
Lchain:
        cp0     [W6]            ;       (*cell) && (cell != &tbl[stop] /*W11*/);
        bra     Z,Lagain        ;
        cp      W6,W11          ;       cell += delta) {
        bra     Z,Lagain        ;   uint16_t adjacent /*W7*/;
Lnoswap:
        mov     [W6+W1],W7      ;   adjacent = *(cell + delta);
        cp      W7,[W6]         ;
        bra     NZ,Lnmatch      ;   if (*cell == adjacent) {
        add     W7,W7,[W6]      ;    *cell += adjacent; // combine cells into 1
        add     W0,[W6],W0      ;    points += *cell; // increase points by same
Lfollow:
        mov     W11,W9          ;    for (uint16_t* tail /*W9*/ = &tbl[stop];
        clr     W10             ;                    tail > cell; end -= delta) {
Lnext:
        mov     [W9],W7         ;     *tail = (tail == &tbl[stop]) ? 0 /*W10*/ :
        mov     W10,[W9]        ;                         *(tail + delta) /*W7*/;
        mov     W7,W10          ;
        sub     W9,W1,W9        ;
        cp      W9,W6           ;    }
        bra     NZ,Lnext        ;   }
Lnmatch:
        add     W6,W1,W6        ;  }
        bra     Lchain          ; // second pass complete: row/column collapsed
Lagain:
        add     W2,W3,W3        ;  start += incr;
        add     W2,W4,W4        ;  stop += incr;
        cp      W4,W5           ;
        bra     LT,L2pass       ; } while (stop < fourth); // board is collapsed
Lscored:
.if 0
 push W0
 mov W0,W2
 mov Ltblsiz,W1
 mov W8,W0
 call drawbrd
 pop W0
.endif
        pop.d   W10             ;
        pop.d   W8              ; return points;
        return                  ;} // apply()

drop:
        ;; upon return: if W2=0,
        ;; table is cleared with
        ;; two tiles set to W3.
        ;; otherwise (if W2>0)
        ;; table contains one
        ;; additional cell at a
        ;; non-obvious, formerly
        ;; zero-valued location
        ;; determined by W3
        add     W0,W1,W1        ;void drop(uint16_t* tbl /*W0->W4*/,
        cp0     W2              ;          uint16_t size /*W1->W5*/,
        bra     NZ,Laddone      ;          uint16_t zeroct /*W2*/,
Lblank:
        mov     W2,[--W1]       ;          uint16_t seed /*W3*/) {
        cp      W1,W0           ; if (zeroct == 0) { // blank the board
        bra     GT,Lblank       ;  uint16_t* i /*W0*/;
        mov     W3,[W0+2]       ;  for (i = &tbl[size]; i > tbl;) *--i = 0;
        mov     W3,[W0+2*COLS]  ;  tbl[1] = tbl[COLS] = seed;
        return                  ; } else { // zeroct is the count of empty cells
Laddone:
        ff1l    W3,W4           ;  uint16_t chosen0 /*W1*/, newval /*W0*/;
        ff1r    W3,W5           ;  uint16_t* i /*W5*/;
        subr    W5,#16,W5       ;
        xor     W4,W5,W3        ;
        mov.d   W0,W4           ;
        and     W3,#0x000f,W3   ;  seed = (ff1L(seed) ^ ff1R(seed)) & 0xf; //<16
        repeat  #17             ;
        div.s   W3,W2           ;  chosen0 = seed % zeroct; //<15 (max. zeroct)
        mov     #0xaaaa,W0      ;
        lsr     W0,W1,W0        ;
        and     W0,#0x0006,W0   ;  newval = (0xaaaa >> chosen0) & 0x0006; //2|4
Lfind0:
        cp0     [--W5]          ;  for (i = &tbl[size]; i > tbl;)
        bra     Z,Lfound0       ;   if (*--i == 0) {
;       cp      W5,W4           ;    if (chosen0)
        bra     GT,Lfind0       ;     chosen0--;
;       bra     $               ;    else {
Lfound0:
        cp0     W1              ;     *i = newval;
        bra     Z,Laddval       ;     return;
        dec     W1,W1           ;    }
        bra     Lfind0          ;   }
Laddval:
        mov     W0,[W5]         ; } // should not reach here: will always find 0
        return                  ;} // drop()

validat:
        ;; upon return: W0 holds
        ;; original value (if 2
        ;; or 8) otherwise 0
        cp      W0,#2           ;uint16_t validat(uint16_t callback_result /*W0*/)
{
        bra     Z,Lisvalid      ; switch (callback_result) {
        cp      W0,#8           ;  case 2: case 8: return W0;
        bra     Z,Lisvalid      ;  default: return 0;
        clr     W0              ;}
Lisvalid:
        return                  ;}

getmove:
        mov     Lgetptr,W2      ;int16_t getmove(uint16_t nretry /*W0*/,
        call    W2              ;                int16_t prmove /*W1*/) {
        btss    W0,#15          ; int16_t retval /*W0*/ = (*Lgetptr)(nretry, prmove
);
        bra     validat         ; if (retval >= 0) return validat(retval); //0/2/8
.ifdef AUTOMOVE
        mov     W0,W1           ; else if (retval == -retval)
```

```
.endif
        neg     W0,W0           ;
.ifdef AUTOMOVE
        cpseq   W0,W1           ;  return retval; // 0x8000 (i.e. jump to autoplay)
.endif
        call    validat         ; else
        neg     W0,W0           ;  return -validat(-retval); // 0/-2/-8
        return                  ;} // getmove()

drawbrd:
        mov     Lputptr,W3      ;void drawbrd(uint16_t* tbl, uint16_t size,
        call    W3              ; uint16_t score) {(*Lputptr)(tbl, size, score);
        return                  ;} // drawbrd()

congrat:
        clr     W0              ;void congrat(void) {
        clr     W1              ;
        mov     Lgoalpwr,W2     ; // callee must recognize null tbl as a "win"
        mov     Lputptr,W3      ;
        call    W3              ; (*Lputptr)((void*)0, 0, Lgoalpwr);
        return                  ;} // congrat()

turn:
        push.d  W8              ;typedef struct { uint16_t p, int16_t m } turn_t;
        push.d  W10             ;turn_t turn(uint16_t* tbl /*W0->W8*/,
        mov.d   W0,W8           ;            uint16_t size /*W1*/,
        mov     #0xffff,W10     ;            int16_t prmove /*W2*/) {
        mov     W2,W11          ; uint16_t points /*W10*/ = 0xffff;
Lloop:
        mov.d   W8,W0           ; int16_t delta /*W11,-8|-2|2|8 drlu*/ = prmove;
;       call    empties         ; do {
;       cp0     W0              ;  if (empties(tbl, siz) == 0))
;       bra     Z,Lbreak        ;   return 0; // no valid moves left MAYBE/FIXME
        mov.d   W10,W0          ;
        call    getmove         ;  delta = getmove(points, delta);
        ior     W0,W0,W11       ;  if (delta == 0)
        bra     Z,Lbreak        ;   return 0; // user requested exit
.ifdef AUTOMOVE
        sl      W11,W0          ;
        bra     NZ,Lapply       ;  if (delta == 0x8000) // user started autoplay
        mov.d   W8,W0           ;
        call    autoplay        ;
        bclr    W0,#0           ;
        bra     Lbreak          ;   return autoplay(tbl, size) & 0xfffe;
Lapply:
.endif
        mov     W8,W0           ;  else
        mov     W11,W1          ;
        call    apply           ;
        mov     W0,W10          ;   points = apply(tbl, delta);
        cp0     W0              ;
        bra     Z,Lloop         ; } while (points == 0); // invalid move entered
Lbreak:
        mov     W11,W1          ;
        pop.d   W10             ;
        pop.d   W8              ; return (turn_t) { points, delta };
        return                  ;} // turn()

iocallbacks:
        mov.d   W0,W2           ;io_t iocallbacks(int16_t (*iptr)() /*W0*/,
        mov     #Lioptrs,W4     ;                 void (*optr)() /*W1*/) {
        mov.d   [W4],W0         ; io_t retval = { Lgetptr, Lputptr };
        clr     W5              ;
        cpsne   W2,W5           ; if (i)
        mov     W0,W2           ;  Lgetptr = i;
        cpsne   W3,W5           ; if (o)
        mov     W1,W3           ;  Lputptr = o;
        mov.d   W2,[W4]         ; return retval;
        return                  ;} // iocallbacks()

        .global subgame
subgame:
        push.d  W8              ;uint16_t subgame(uint16_t* tbl /*W0->W10*/,
        push.d  W10             ;                 uint16_t size /*W1->W11*/) {
        push    W12             ;
        clr     W12             ; uint16_t score /*W12*/ = 0; // odd if won yet
        mov     #0x8000,W9      ; turn_t points_prmove /*W8W9*/; // { .p, .m }
        mov.d   W0,W10          ; points_prmove.m = 0x8000; // initial move
Lgamelp:
        mov.d   W10,W0          ; do {
        mov     W12,W2          ;
        bclr    W2,#0           ;
        call    drawbrd         ;  drawbrd(Lboard, Ltblsiz, score & 0xfffe);
        mov.d   W10,W0          ;
        mov     W9,W2           ;
        call    turn            ;  points_prmove = turn(Lboard, Ltblsiz,
        mov.d   W0,W8           ;   points_prmove.m); // points odd means tilted
        cp0     W0              ;  if (score + points < score)
        bra     Z,Lgameovr      ;   points = 0; // handle overflow as abort/lost
        bclr    W0,#0           ;  else { // game not over, aborted or score wrap
        add     W12,W0,W0       ;   uint16_t zeroct /*W0*/;
        bra     C,Lgameovr      ;
        mov     W0,W12          ;   score += points & 0xfffe;
        mov.d   W10,W0          ;
        call    empties         ;   zeroct = empties(Lboard, Ltblsiz);
        mov     W0,W2           ;
        xor     W0,W12,W3       ;
        mov.d   W10,W0          ;
        call    drop            ;   drop(Lboard, Ltblsiz, zeroct, score^zeroct);
        btsc    W12,#0          ;
        bra     Lgamelp         ;   if ((score & 0x0001 == 0) && // not won yet
        mov.d   W10,W0          ;
        mov     Lgoalpwr,W2     ;
        call    gamewon         ;       gamewon(Lboard, Ltblsiz, Lgoalpwr)) {
        ior     W0,W12,W12      ;    score |= 0x0001;
        btsc    W0,#0           ;    congrat(); // only call congrat() once
        call    congrat         ;   }
        bra     Lgamelp         ;  }
Lgameovr:                       ;
        mov     W12,W0          ; } while (points);
        pop     W12             ;
        pop.d   W10             ;
        pop.d   W8              ; return score;
        return                  ;} // subgame()

        .global newgame
newgame:
        push.d  W8              ;void newgame(uint16_t rows /*W0*/,
        mov.d   W2,W8           ;             uint16_t goal /*W1*/,
        call    init            ;             int16_t (*getcb)(void) /*W2*/,
        mov     #Lboard,W0      ;             void (*putcb)(...) /*W3*/) {
        mov     Ltblsiz,W1      ; init(rows, goal);
        mov     #0,W2           ;
        mov     #2,W3           ;
        call    drop            ; drop(Lboard, Ltblsiz, 0, 2); // 2s abut corner
        mov.d   W8,W0           ;
        pop.d   W8              ;
        call    iocallbacks     ; (void) iocallbacks(getcb, putcb);
        return                  ;} // newgame()

        .global game
game:
        call    newgame         ;uint16_t game(uint16_t rows, uint16_t goal,
        mov     #Lboard,W0      ;         int16_t (*getcb)(), void (*putcb)()) {
        mov     Ltblsiz,W1      ; newgame (rows, goal, getcb, putcb);
        call    subgame         ; return subgame(Lboard, Ltblsiz);
        return                  ;} // game()
```

```
.ifdef AUTOMOVE
autopress:
        sl      W1,W2           ;int16_t autopress(uint16_t nretry /*W0*/,
        bra     NZ,Lnfirst      ;                  int16_t prmove /*W1*/) {
        mov     #2,W1           ; if (prmove == 0x8000) // first call by turn()
Lnfirst:
        clr     W2              ;  prmove = 2; // spoof a left to get an up
        cpseq   W0,W2           ;
        xor     W1,#0xa,W0      ; return nretry ? (prmove ^ (8|2)) : 0;
        return                  ;} // autopress() // ULULUL until stuck

autoplay:
        push.d  W8              ;uint16_t autoplay(uint16_t* tbl /*W0*/,
        mov.d   W0,W8           ;                  uint16_t size /*W1*/) {
       mov #handle(autopress),W0; uint16_t autopoints;
        clr     W1              ; io_t oldio /*W8W9*/;
        call    iocallbacks     ; oldio = iocallbacks(autopress, (void*)0);
        exch    W8,W0           ; // W0 is now tbl, W8 is storing oldio.i
        mov     W9,W1           ; // W1 is now size
        call    subgame         ; autopoints = subgame(tbl, size);
        exch    W8,W0           ; // W0 is now oldio.i, W8 is storing autopoints
        clr     W1              ;
        call    iocallbacks     ; (void) iocallbacks(oldio.i, (void*)0);
        mov     W8,W0           ;
        pop.d   W8              ; return autopoints;
        return                  ;} // autoplay()

        .global autogame
autogame:
        call    newgame         ;uint16_t game(uint16_t rows, uint16_t goal,
        mov     #Lboard,W0      ;         int16_t (*getcb)(), void (*putcb)()) {
        mov     Ltblsiz,W1      ; newgame (rows, goal, getcb, putcb);
        call    autoplay        ; return subgame(Lboard, Ltblsiz);
        return                  ;} // game()
.endif
        .end

// template to start a 4x4 tile game in a program (stdio-based example):
#include <stdio.h>

int16_t vi_input(void) {
 do switch (getchar()) {
  case 'h': return 2;
  case 'j': return -8;
  case 'k': return 8;
  case 'l': return -2;
  case 'a': return 0x8000;
  case '\033': return 0;
 } while (1);
}

void simple_output(uint16_t* grid, uint16_t gridbytes, uint16_t value) {
 if (grid) {
  printf("\nYour score: %d\n", value);
  for (int i = 0; i < gridbytes>>3; i++) {
   for (int j = 0; j < 4; j++)
    if ((value = *grid++) > 0) printf("%-5d", value); else printf("     ");
   printf("\n");
  }
 } else printf("\nCongratulations! You reached the %d tile.\n", value);
}

uint16_t play2048 (void) { return game(4, 2048, vi_input, simple_output); }
```

```
#include <stdint.h>
#include "lcd.h"

// for proper \n => \r\n behavior, need outString()
#include <pic24_all.h>

void printlcd(char const* str) {
  uint16_t i;
  for (i = 0; str[i]; i++)
   LCD_PutChar(str[i]);
  outString(str);
}

void printlcd4d(uint16_t val) {
 char num[5];
 uint16_t i;

 num[0] = ' ';
 num[1] = ' ';
 num[2] = ' ';
 num[3] = '0';
 num[4] = '\0';

 for (i = 3; val > 0; i--) {
  uint16_t dig;

  if (i < 0) {
   num[0] = 'O';
   num[1] = 'v';
   num[2] = 'r';
   num[3] = 'f';
   break;
  }
  dig = val % 10;
  val = val / 10;
  num[i] = ((char) dig) + '0';
 }

 printlcd(num);
}

void osimple(uint16_t* grid, uint16_t gridbytes, uint16_t value) {
 uint16_t i, j;

 if (grid) {
  for (i = 0; i < gridbytes>>3; i++) {
   outString("\n");
   for (j = 0; j < 4; j++)
    if ((value = *grid++) > 0)
     printlcd4d(value);
    else
     printlcd("[__]");
  }
 } else {
  printlcd("Congratulations!You reached ");
  printlcd4d(value);
 }
 outString("\n");
}

uint8 spress(void);

void summary(uint16_t score) {
 if (score & 1)
  printlcd("\nYou won with a good score: ");
 else
  printlcd("\nYou fell short. You scored ");

 printlcd4d(score & 0xfffe);
 printlcd(".");
 outString("\n");
}

void ssetup(void) {
 CONFIG_RD6_AS_DIG_INPUT(); // S3 (H)
 CONFIG_RD7_AS_DIG_INPUT(); // S6 (J)
 CONFIG_RA7_AS_DIG_INPUT(); // S5 (K)
 CONFIG_RD13_AS_DIG_INPUT(); // S4 (L)
}

uint8 S3pressed(void) {
  return (PORTDbits.RD6 == 0) ? 1 : 0;
}

uint8 S6pressed(void) {
  return (PORTDbits.RD7 == 0) ? 1 : 0;
}

uint8 S5pressed(void) {
  return (PORTAbits.RA7 == 0) ? 1 : 0;
}

uint8 S4pressed(void) {
  return (PORTDbits.RD13 == 0) ? 1 : 0;
}

uint8 dbounc(uint8 (*f)(void), uint8 c) {
  uint16 b = 0xffff;

  do {
    b <<= 1;
    b |= (*f)();
    doHeartbeat();
  } while (b);
  return c;
}

uint8 spress(void) {
 doHeartbeat();
 if (S3pressed()) return dbounc(S3pressed, 'h');
 if (S6pressed()) return dbounc(S6pressed, 'j');
 if (S5pressed()) return dbounc(S5pressed, 'k');
 if (S4pressed()) return dbounc(S4pressed, 'l');
 return 0;
}
```

```
.include "p24FJ128GA010.inc"

        .section psv psv
HELLO_MSG:
        .asciz  "2048-style game using vi keys HJKL to move, Esc to exit\n"

        .text
exp16inp:
        ;; read tact switches here
        return

vi_input:
        call    _inChar         ;int16_t vi_input (void) {
        and     W0,#0x1f,W0     ; do switch (0x1f & getchar()) {
        cp      W0,#0x01        ; case 'a' - '':
        bra     NZ,Lleft        ;
.ifdef AUTOMOVE
        mov     #0x8000,W0      ;
        return                  ;  return 0x8000;
.endif
Lleft:
        cp      W0,#0x08        ; case 'h' - '':
        bra     NZ,Ldown        ;
        mov     #0x0002,W0      ;
        return                  ;  return +2; // left
Ldown:
        cp      W0,#0x0a        ; case 'j' - '':
        bra     NZ,Lup          ;
        mov     #0xfff8,W0      ;
        return                  ;  return -8; // down
Lup:
        cp      W0,#0x0b        ; case 'k' - '':
        bra     NZ,Lright       ;
        mov     #0x0008,W0      ;
        return                  ;  return +8; // up
Lright:
        cp      W0,#0x0c        ; case 'l' - '':
        bra     NZ,Lesc         ;
        mov     #0xfffe,W0      ;
        return                  ;  return -2; // right
Lesc:
        cp      W0,#0x1b        ; case '\033':
        bra     NZ,vi_input     ;  return 0; // quit
        clr     W0              ; } while (1);
        return                  ;}

.global _main
_main:
        mov     #HELLO_MSG,W0   ;
        call    _configBasic    ;
        call    _ssetup         ;
Lgamelp:
        .equ TWOROWLCD,2
.ifdef TWOROWLCD
        call    _LCD_Initialize ;
        mov     #TWOROWLCD,W0   ;
        mov     #256,W1         ;
.else
        mov     #4,W0           ;
        mov     #2048,W1        ;
.endif

.if 0
 mov #handle(vi_input),W2
 mov #handle(nullfunc),W3
 call autogame
.else
        mov #handle(vi_input),W2;
        mov #handle(_osimple),W3;
        call    game            ;
        call    _summary        ;
.endif
        goto    $
nullfunc:
        return
        .end
```