Sample solution:

User interface:

The user interface defines what a user has to input and what is expected as the output.

INPUT: In the design presented here, each entry consists of three one-digit numbers (i, j, d), denoting row number, column number and the number to be put in the Sudoku grid. The rows are numbered 0 to 8 from top to bottom and the columns are numbered 0 to 8 from left to right. A value d = 0 is used to indicate that it is the last entry, for file input as well as for key-board input. The numbers in the file (initial entries) are read as integers and these need to be separated by space (s) and/or new line character (s). Numbers from the key-board (subsequent entries) are read as characters and do not require any separator.

OUTPUT: Since the LCD display has only 15 rows, we display only the block boundaries as shown, using only 13 rows. All the messages are displayed on Stdout, including prompts for key-board input.

```
 ---  ---  ---
|    |    |    |
|    |    |    |
|    |    |    |
 ---  ---  ---
|    |    |    |
|    |    |    |
|    |    |    |
 ---  ---  ---
|    |    |    |
|    |    |    |
|    |    |    |
 ---  ---  ---
```

Overall design at high level:

- Start with displaying empty grid. Read the initial entries from a file, keeping track of the number of entries and checking forconflicts, till the terminating entry is read (d = 0). Only conflict free numbers entered are displayed in the grid.
- Get subsequent entries from the key-board, keeping track of the number of entries, checking for conflicts and displaying conflict free numbers in the grid. End the program either when the grid is full (numbers entered reach 81), declaring “success”, or when the terminating entry is read (d = 0) before the grid is full, declaring “failure”.
- While checking for conflicts, appropriate messages are displayed if row conflicts or column conflicts or block conflicts are found.

The main program is as follows.

```
int main ( ) {
        int n, i, j, d;
        n = initialize ( );             /* Fill and display initial entries, return the number of entries */
        do {
                if (n == 81) {message (“success”); exit;};                              /* Grid completed */
                message (“enter row, column and digit”);                     /* Prompt for key-board input */
                i = get (8); j = get (8); d = get (9);             /* Range for i, j = 0..8, range for d = 0..9 */
                if (d == 0) {message (“failure”); exit;};                                    /* User quits */
                if (check (i, j, d) == 0) {                                         /* Check for conflicts */
                        update (i, j, d);                                 /* Update the grid if no conflicts */
                        n++;
                }
        };
};
```

The initialization function contains a loop similar to the loop in main, except that input comes from a file here rather than key-board.

```
int initialize ( ) {
        int i, j, d, f; int n = 0;
        clear_grid ( );                                             /* Set all grid entries to 0. */
        display_grid ( );                                              /* Display empty grid */
        f = open_file ("Sudoku.txt");                 /* Open file "Sudoku.txt" and get file handle f */
        do {                                               /* Range for i, j = 0..8, range for d = 0..9 */
                i = read_file (f, 8); j = read_file (f, 8); d = read_file (f, 9);   /* Get i, j and d from f */
                if (d == 0) exit;                                          /* Last entry in the file */
                if (check (i, j, d) == 0) {                                /* Check for conflicts */
                        update (i, j, d);                          /* Update the grid if no conflicts */
                        n++;
                }
        };
        close_file (f);                                                            /* close file f */
        return (n);
};
```

The grid is a 9x9 array.

```
int grid [9] [9];
```

Conflicts for an entry (i, j, d) are checked by looking at the existing entries in the array. The function "check" calls three functions, one for row check, one for column check and one for block check. These functions return "1" if there is a conflict and return "0" otherwise. The checks required are same whether the input comes from a file or from the key-board.

```
int check (int i, j, d) {
        return (check_row (i, d) + check_col (j, d) + check_block (i, j, d));
};
--------------------------------------------------------------------
int check_row (int i, int d) {
        for (jj = 0; jj < 9; jj++)                       /* Check if d is already present in row i */
                if (grid [i][jj] == d) {message ("row conflict"); return (1); };
        return (0);
};
--------------------------------------------------------------------
int check_col (int j, int d) {
        for (ii = 0; ii < 9; ii++)                    /* Check if d is already present in column j */
                if grid [ii][j] == d) {message ("column conflict"); return (1); };
        return (0);
};
--------------------------------------------------------------------
int check_block (int i, int j, int d) {
        int block [9] = (0, 0, 0, 3, 3, 3, 6, 6, 6);
                          /* This array is used to look-up the starting position of the current block */
                    /* block [i] and block [j] give the top left corner of the block containing cell i, j */
        for (ii = block [i]; ii < block [i] + 3; ii++)       /* Check if d is already present in the block */
                for (jj = block [j]; jj < block [j] + 3; jj++)
                        if (grid [ii][jj] == d) {message ("block conflict"); return (1); };
        return (0);
};
--------------------------------------------------------------------
```

The function “clear_grid” initialize all grid cells to 0.

```
void clear_grid ( ) {
        for (i = 0; i < 9; i++)
                for (j = 0; j < 9; j++) grid [i][j] = 0;
};
```

The function “update” enters digit d in the cell i, j of the grid. It also displays the digit at appropriate position. It is called after conflict check.

```
void update (int i, j, d) {
        int place [9] = (1,2,3,5,6,7,9,10,11);
                                /* This array is used to look-up the position where d is to be displayed */
                                                /* It takes into account the position of grid lines. */
        grid [i][j] = d;
        display_character (place [j], place [i], d + ’0’);
                        /* Display character representing digit d at column: place [j], row: place [i]. */
};
```

The grid is displayed using two patterns, defined as line [0] and line [1]. The index array is looked-up to find which pattern is to be displayed in which row.

```
void display_grid ( ) {
        char * line [2] = (“  ---  ---  ---  ”, “|    |    |    |”);
        int index [13] = (0,1,1,1,0,1,1,1,0,1,1,1,0)
        for (i = 0; i < 13; i++) display_line (i, line [index [i]]);
};
```

The function “get_key” keeps reading blue buttons till a non-zero value is returned (indicates that a blue button has been pressed). The returned value is matched with the patterns corresponding to digits 0 to m to determine which key is pressed. Keys other than 0 to m are ignored.

```
int pattern [10] = (1,2,4,8,16,32,64,128,256,512); /* pattern [i] has ith bit = 1 and other bits = 0. */
int get_key (int m) {
        do {
                do {k = blue_key ( ); }; while (k == 0);
                for (i = 0; i <= m; i++) if (k == pattern [i]) return (i);
        };
};
```

The function “read_file” reads an integer value from the file with handle f and checks if it lies in the range 0 .. m. If the value is out of range, it repeats the process.

```
int read_file (int f, m) {
        do {
                k = read_int (f);
                if (k >= 0 && k <= m) return (k);
        };
};
```

Some names have been shortened in the assembly program –

initialize => init,	check_row => checkr,	check_col => checkc,	check_block => checkb
clear_grid => clear,	display_grid => dispg,	get_key => get,	read_file => readf

Assembly code

File sudoku1.s

```
                                              |         .data
char * m1 = “row conflict”;                   | m1:     .asciz "Row Conflict\n"
char * m2 = “column conflict”;                | m2:     .asciz "Column Conflict\n"
char * m3 = “block conflict”;                 | m3:     .asciz "Block Conflict\n"
                                              |         .text
                                              |         .global chkr, chkc, chkb
----------------------------------------------+------------------------------------------------------------
int chkr (int *p, int d) {int t; int *q;      | chkr:   str lr, [sp, # -4]!      @ p, d, t, q : r0, r1, r2, r3
        q = p + 9;                            |         add r3, r0, # 9
        do {    t = * p;                      | loop1:  ldrb r2, [r0]
                if (t == d) {                 |         cmp r2, r1
                                              |         bne L1
                        message (m1);         |         ldr r0, = m1             @ row conflict found
                                              |         swi 0x02                 @ display message
                        return (1);           |         mov r0, # 1
                };                            |         b ret1
                p++;                          | L1:     add r0, r0, # 1
        } while p < q;                        |         cmp r0, r3
                                              |         blt loop1                @ row not yet over
        return (0);                           |         mov r0, # 0
};                                            | ret1:   ldr pc, [sp], # 4
----------------------------------------------+------------------------------------------------------------
int chkc (int * p, int d) {int t; int *q;     | chkc:   str lr, [sp, # -4]!      @ p, d, t, q : r0, r1, r2, r3
        q = p + 81;                           |         add r3, r0, # 81
        do {    t = * p;                      | loop2:  ldrb r2, [r0]
                if (t == d) {                 |         cmp r2, r1
                                              |         bne L2
                        message (m2);         |         ldr r0, = m2             @ column conflict found
                                              |         swi 0x02                 @ display message
                        return (1);           |         mov r0, # 1
                };                            |         b ret2
                p += 9;                       | L2:     add r0, r0, # 9
        } while p < q;                        |         cmp r0, r3
                                              |         blt loop2                 @ column not yet over
        return (0);                           |         mov r0, # 0
};                                            | ret2:   ldr pc, [sp], # 4
----------------------------------------------+------------------------------------------------------------
int chkb (int*p,int d){int t;int*q1;int *q2;  | chkb:   str lr, [sp, # -4]!    @ p,d,t,q1,q2: r0,r1,r2,r3,r4
                                              |         str r4, [sp, # -4]!       @ callee save
        q1 = p + 3;                           |         add r3, r0, # 3           @ to check end of row
        q2 = p + 27;                          |         add r4, r0, # 27          @ to check end of block
        do {    do {    t = * p;              | loop3:  ldrb r2, [r0]
                        if (t == d) {         |         cmp r2, r1
                                              |         bne L3
                           message (m3);      |         ldr r0, = m3                @ block conflict found
                                              |         swi 0x02                    @ display message
                           return (1);        |         mov r0, # 1
                        };                    |         b ret3
                        p ++;                 | L3:     add r0, r0, # 1
                } while p < q1;               |         cmp r0, r3
                                              |         blt loop3                    @ row not yet over
                p += 6;                       |         add r0, r0, # 6
                q1 += 9;                      |         add r3, r3, # 9
        } while p < q2;                       |         cmp r0, r4
                                              |         blt loop3                     @ block not yet over
        return (0);                           |         mov r0, # 0
};                                            | ret3:   ldr r4, [sp], # 4              @ callee restore
                                              |         ldr pc, [sp], # 4
----------------------------------------------+------------------------------------------------------------
                                              |         .end
```

File sudoku2.s

```
                                          .data
int block [9] = (0,0,0,3,3,3,6,6,6);  block:   .byte 0,0,0,3,3,3,6,6,6        @ block position
int place [9] = (1,2,3,5,6,7,9,10,11); place:   .byte 1,2,3,5,6,7,9,10,11   @ location for display
                                          .text
                                          .global check, clear, update
                                          .extern chkr, chkc, chkb
```

```
void clear ( ) { p = g;               clear:   mov r0, r5               @ r5 has &grid [ ][ ]
        i = 0;                                 mov r1, # 0
        q = g + 81;                            add r2, r0, # 81
        do { *p++ = i;                loop:    strb r1, [r0], # 1
        } while p < q;                         cmp r0, r2
                                               blt loop
};                                             mov pc, lr
```

```
int check (int i, j, d) {             check:   str lr, [sp, # -4]!      @ i, j, d : r0, r1, r2
        int * p, q;                            str r4, [sp, # -4]!      @ callee saved
        int t, k;                              str r2, [sp, # -4]!      @ caller saved
                                               str r1, [sp, # -4]!      @ caller saved
                                               str r0, [sp, # -4]!      @ caller saved
        conflict = 0;                          mov r4, # 0              @ conflict : r4
        t = i * 9;                             add r0, r0, r0, LSL # 3  @ t : r0
        p = g + t;                             add r0, r5, r0           @ r5 has &grid [ ][ ]
        t = chkr (p, d);                       mov r1, r2
                                               bl chkr                  @ call for row check
        conflict += t;                         add r4, r4, r0
        p = g + j;                             ldr r1, [sp, # 4]        @ j : from stack in r1
                                               add r0, r5, r1           @ r5 has &grid [ ][ ]
        t = chkc (p, d);                       ldr r1, [sp, # 8]        @ d : from stack in r1
                                               bl chkc                  @ call for column check
        conflict += t;                         add r4, r4, r0
        q = &block [0];                        ldr r3, = block          @ q : r3
        t = *(q + i);                          ldr r0, [sp], # 4        @ i : from stack in r0
                                               ldrb r0, [r3, r0]
        t = t * 9                              add r0, r0, r0, LSL # 3
        p = g + t;                             add r0, r5, r0           @ r5 has &grid [ ][ ]
        k = *(q + j)                           ldr r1, [sp], # 4        @ j : from stack in r1
                                               ldrb r1, [r3, r1]        @ k : r1
        p = p + k;                             add r0, r0, r1
        t = chkb (p, d);                       ldr r1, [sp], # 4        @ d : from stack in r1
                                               bl chkb                  @ call for block check
        conflict += t;                         add r0, r4, r0
        return (conflict);                     ldr r4, [sp], # 4        @ callee restored
};                                             ldr pc, [sp], # 4
```

```
void update (int i, j, d) {           update:  str lr, [sp, # -4]!      @ i, j, d : r0, r1, r2
        int * p, q;                            str r4, [sp, # -4]!      @ callee saved
        int t;
        t = i * 9;                             add r4, r0, r0, LSL # 3  @ t : r4
        p = g + t;                             add r4, r5, r4           @ r5 has &grid [ ][ ]
        p = p + j;                             add r4, r4, r1
        * p = d;                               strb r2, [r4]
        q = &place [0]                         ldr r3, = place          @ q : r3
        t = *(q + i);                          ldrb r4, [r3, r0]        @ t : r4
        i = *(q + j);                          ldrb r0, [r3, r1]
        j = t;                                 mov r1, r4
        dispch (j, i, d+'0'));                 add r2, r2, # 48
                                               swi 0x207               @ display a character
};                                             ldr r4, [sp], # 4        @ callee restored
                                               ldr pc, [sp], # 4
```

```
                                               .end
```

File sudoku3.s

int pat [10] = (1,2,4,8,16,32,64,128,256,512);
char * line0 = " --- --- --- ";
char * line1 = "| | | |";
int index [13] =
  (0,14,14,14,0,14,14,14,0,14,14,14,0)
	.data
	pat: .word 1,2,4,8,16,32,64,128,256,512
	line0: .asciz " --- --- --- "
	line1: .asciz "| | | |"
	index: .byte 0,14,14,14,0,14,14,14,0,14,14,14,0
	   .text
	   .global dispg, get, readf

void dispg ( ) {
 	
  i = 0;
  do { p = &index [0];
    k = *(p + i);
    p = &line0 [0];
    p = p + k;
    displine (i, p);
    i++;
  } while (i < 13);
 	
};
	dispg: str lr, [sp, # -4]!
	   mov r0, # 0 @ column no. for display
	   mov r1, # 0 @ row no. for display
	loop: ldr r2, = index
	   ldrb r3, [r2, r1] @ line0 or line 1
	   ldr r2, = line0
	   add r2, r2, r3 @ offset 14 for line1
	   swi 0x204 @ display a line
	   add r1, r1, # 1
	   cmp r1, # 13
	   blt loop
	   ldr pc, [sp], # 4

int get (int m) {
 	
  do { do { k = blue_key ( );
    } while k == 0;
 	
    i = 0;
    do {
      p = pat [i];
      if (k == p)
        return (i);
      i++;
    } while (i <= m)
 	
  };
};
	get: str lr, [sp, # -4]!
	   mov r3, r0
	L1: swi 0x203 @ get blue key
	   cmp r0, # 0 @ is key pressed?
	   beq L1 @ wait for key
	   mov r2, # 0 @ i : r2
	L2: ldr r1, = pat
	   ldr r1, [r1, r2, LSL # 2]
	   cmp r0, r1 @ match key pattern
	   beq ret
	   add r2, r2, # 1
	   cmp r2, r3
	   ble L2
	   b L1 @ try again
	ret: mov r0, r2 @ return i
	   ldr pc, [sp], # 4

int readf (int f, m) {
  do {
    k = read_int (f);
    if (k >= 0
 	
      && k <= m)
 	
      return (k);
  };
};
	readf: str lr, [sp, # -4]!
	L3:
	   swi 0x6c @ read integer
	   cmp r0, # 0
	   blt back @ less than zero
	   cmp r0, r1
	   bgt back @ greater than m
	   ldr pc, [sp], # 4
	back: b L3 @ try again

	.end

File sudoku4.s

```
                                            .data
char * filename = “Sudoku.txt”;     file:   .asciz "Sudoku.txt"
                                            .text
                                            .global init
                                            .extern clear, dispg, readf, check, update

int init ( ) {                      init:   str lr, [sp, # -4]!
    int i, j, d;                            str r6, [sp, # -4]!       @ callee saved
                                            str r7, [sp, # -4]!       @ callee saved
    int n = 0;                              mov r6, # 0               @ n : r6
    clear ( );                              bl clear                  @ clear the grid
    dispg ( );                              bl dispg                  @ display blank grid
    h = open_file (filename);               mov r1, # 0
                                            ldr r0, = file
                                            swi 0x66                  @ open file
                                            mov r7, r0                @ h : r7
    do {    i = readf (h, 8);       L:      mov r0, r7                @ read i (range 0..8)
                                            mov r1, # 8
                                            bl readf
                                            str r0, [sp, # -4]!       @ push i
            j = readf (h, 8);               mov r0, r7                @ read j (range 0..8)
                                            mov r1, # 8
                                            bl readf
                                            str r0, [sp, # -4]!       @ push j
            d = readf (h, 9);               mov r0, r7                @ read d (range 0..9)
                                            mov r1, # 9
                                            bl readf
                                            str r0, [sp, # -4]!       @ push d
            if (d == 0) exit;               cmp r0, # 0
                                            addeq sp, sp, # 12        @ adjust stack
                                            beq exit
            t = check (i, j, d);            mov r2, r0
                                            ldr r1, [sp, # 4]         @ j from stack
                                            ldr r0, [sp, # 8]         @ i from stack
                                            bl check
            if (t == 0) {                   cmp r0, # 0
                                            addne sp, sp, # 12        @ adjust stack
                                            bne L
                update (i, j, d);           ldr r2, [sp], # 4         @ pop d from stack
                                            ldr r1, [sp], # 4         @ pop j from stack
                                            ldr r0, [sp], # 4         @ pop i from stack
                                            bl update
                n++;                        add r6, r6, # 1
            };                              b L
    };
    close_file (h);                 exit:   mov r0, r7
                                            swi 0x68                  @ close file
    return (n);                             mov r0, r6
};                                          ldr r7, [sp], # 4         @ callee restored
                                            ldr r6, [sp], # 4         @ callee restored
                                            ldr pc, [sp], # 4

                                            .end
```

File sudoku5.s

int * g;		.data	
int grid [9] [9];	grid:	.space 81	
char * m1 = “Success”;	m1:	.asciz "Success"	
char * m2 = “Failure”;	m2:	.asciz "Failure"	
char * m3 = “enter row, column and digit”;	m3:	.asciz "enter row column and digit\n"	
		.text	
		.global _start	
		.extern init, get, check, update	
int main ( ) { int n, i, j, d;	_start:		
g = & grid [0][0];		ldr r5, = grid	@ used as global
n = init ( );		bl init	@ initialize
		mov r6, r0	@ n : r6
do { if (n == 81) {	L:	cmp r6, # 81	
		bne cont1	
message (m1);		ldr r0, = m1	
		swi 0x02	@ success message
exit;		b exit	
};	cont1:		
message (m3);		ldr r0, = m3	
		swi 0x02	@ prompt for input
i = get (8);		mov r0, # 8	
		bl get	@ get i (range 0..8)
		str r0, [sp, # -4]!	@ push i
j = get (8);		mov r0, # 8	
		bl get	@ get j (range 0..8)
		str r0, [sp, # -4]!	@ push j
d = get (9);		mov r0, # 9	
		bl get	@ get d (range 0..9)
		str r0, [sp, # -4]!	@ push d
if (d == 0) {		cmp r0, # 0	
		bne cont2	
		add sp, sp, # 12	@ adjust stack
message (m2);		ldr r0, = m2	
		swi 0x02	
exit;		b exit	
};			
t = check (i, j, d);	cont2:	mov r2, r0	@ d from r0 to r2
		ldr r1, [sp, # 4]	@ j from stack
		ldr r0, [sp, # 8]	@ i from stack
		bl check	
if (t == 0) {		cmp r0, # 0	
		addne sp, sp, # 12	@ adjust stack
		bne L	
update (i, j, d);		ldr r2, [sp], # 4	@ pop d from stack
		ldr r1, [sp], # 4	@ pop j from stack
		ldr r0, [sp], # 4	@ pop i from stack
		bl update	
n++;		add r6, r6, # 1	
};		b L	
};	exit:	swi 0x11	
};			
		.end	