COL216 Computer Architecture Lab Test 1 10.02.2018

Design a program in ARM assembly language to fill up a SUDOKU puzzle. This puzzle requires digits 1 to 9 to be filled in a 9 x 9 grid, sub-divided into nine 3x3 blocks, such that each row, each column and each block has exactly one occurrence of each digit. The grid has a few numbers initially filled. These initial values are to be read from a file. The numbers filled in subsequently are to be entered from the 4x4 keyboard (blue buttons) of the Embest board plug-in. The grid is to be displayed in the LCD screen of the Embest board plug-in. The messages about successful completion and wrong moves may be displayed on the same LCD screen or on the Stdout. SWI codes are given below. Clearly define your user interface and overall design at high level. Structure your program into multiple files. Each file should not be more than a page in the answer script. Put enough comments in the code to make it easily understandable.

Opcode	Description and Action	Inputs	Outputs	EQU
swi 0x00	Display Character on Stdout	r0: the character		SWI_PrChr
swi 0x02	Display String on Stdout	r0: address of a null terminated ASCII string	(see also 0x69 below)	
swi 0x11	Halt Execution			SWI_Exit
swi 0x66	Open File (mode values in r1 are: 0 for input, 1 for output, 2 for appending)	r0: file name, i.e. address of a null terminated ASCII string containing the name r1: mode	r0:file handle If the file does not open, a result of -1 is returned	SWI_Open
swi 0x68	Close File	r0: file handle		SWI_Close
swi 0x6c	Read Integer from a File	r0: file handle	r0: the integer	SWI_RdInt

Opcode	Description and Action	Inputs	Outputs
swi 0x203	Check if one of the Blue Buttons has been pressed.	None	r0 = the Blue Button Pattern
swi 0x204	Display a string on the LCD screen	r0: x position coordinate on the LCD screen (0-39); r1: y position coordinate on the LCD screen (0-14); r2: Address of a null terminated ASCII string.	The string is displayed starting at the given position of the LCD screen.
swi 0x205	Display an integer on the LCD screen	r0: x position coordinate on the LCD screen (0-39); r1: y position coordinate on the LCD screen (0-14); r2: integer to print.	The string is displayed starting at the given position of the LCD screen.
swi 0x206	Clear the display on the LCD screen	None	Blank LCD screen.
swi 0x207	Display a character on the LCD screen	r0: x position coordinate on the LCD screen (0-39); r1: y position coordinate on the LCD screen (0-14); r2: the character.	The string is displayed starting at the given position of the LCD screen.
swi 0x208	Clear one line in the display on the LCD screen	r0: line number (y coordinate) on the LCD screen	Blank line on the LCD screen.

Marking scheme:

The answer should contain the following three things.

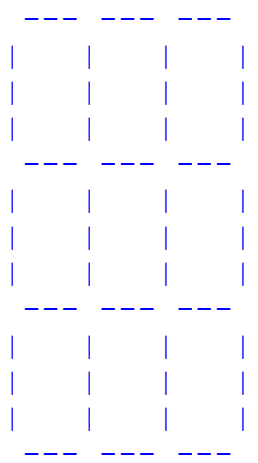
1. User interface [1 mark]
2. Overall design at high level [1 mark]
3. ARM assembly language program
   - 3.1. The program has the following functional requirements
     - 3.1.1. Reading and filling initial entries from a file [1 mark]
     - 3.1.2. Filling subsequent entries from the 4x4 keyboard [1 mark]
     - 3.1.3. Displaying grid entries filled in on the LCD screen [1 mark]
     - 3.1.4. Checking for row and column/ conflicts [1 mark]
     - 3.1.5. Checking for block conflicts [1 mark]
     - 3.1.6. Displaying success/failure messages and wrong moves on the LCD screen or on the Stdout [1 mark]
   - 3.2. The program has the following general requirements
     - 3.2.1. Program is to be structured into multiple files, each file not be more than a page in the answer script [1 mark]
     - 3.2.2. There should be enough comments in the code to make it understandable. [1 mark]

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 i^th 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

char * m1 = “row conflict”; char * m2 = “column conflict”; char * m3 = “block conflict”;	.data m1: .asciz "Row Conflict\n" m2: .asciz "Column Conflict\n" m3: .asciz "Block Conflict\n" .text .global chkr, chkc, chkb

int chkr (int *p, int d) {int t; int *q;
q = p + 9;
do { t = * p;
if (t == d) {
message (m1);
return (1);
};
p++;
} while p < q;
return (0);
};
	chkr: str lr, [sp, # -4]! @ p, d, t, q : r0, r1, r2, r3
	add r3, r0, # 9
	loop1: ldrb r2, [r0]
	cmp r2, r1
	bne L1
	ldr r0, = m1 @ row conflict found
	swi 0x02 @ display message
	mov r0, # 1
	b ret1
	L1: add r0, r0, # 1
	cmp r0, r3
	blt loop1 @ row not yet over
	mov r0, # 0
	ret1: ldr pc, [sp], # 4

int chkc (int * p, int d) {int t; int *q;
q = p + 81;
do { t = * p;
if (t == d) {
message (m2);
return (1);
};
p += 9;
} while p < q;
return (0);
};
	chkc: str lr, [sp, # -4]! @ p, d, t, q : r0, r1, r2, r3
	add r3, r0, # 81
	loop2: ldrb r2, [r0]
	cmp r2, r1
	bne L2
	ldr r0, = m2 @ column conflict found
	swi 0x02 @ display message
	mov r0, # 1
	b ret2
	L2: add r0, r0, # 9
	cmp r0, r3
	blt loop2 @ column not yet over
	mov r0, # 0
	ret2: ldr pc, [sp], # 4

int chkb (int*p,int d){int t;int*q1;int *q2;
q1 = p + 3;
q2 = p + 27;
do { do { t = * p;
if (t == d) {
message (m3);
return (1);
};
p ++;
} while p < q1;
p += 6;
q1 += 9;
} while p < q2;
return (0);
};
	chkb: str lr, [sp, # -4]! @ p,d,t,q1,q2: r0,r1,r2,r3,r4
	str r4, [sp, # -4]! @ callee save
	add r3, r0, # 3 @ to check end of row
	add r4, r0, # 27 @ to check end of block
	loop3: ldrb r2, [r0]
	cmp r2, r1
	bne L3
	ldr r0, = m3 @ block conflict found
	swi 0x02 @ display message
	mov r0, # 1
	b ret3
	L3: add r0, r0, # 1
	cmp r0, r3
	blt loop3 @ row not yet over
	add r0, r0, # 6
	add r3, r3, # 9
	cmp r0, r4
	blt loop3 @ block not yet over
	mov r0, # 0
	ret3: ldr r4, [sp], # 4 @ callee restore
	ldr pc, [sp], # 4

	.end

File sudoku2.s

```
int block [9] = (0,0,0,3,3,3,6,6,6);
int place [9] = (1,2,3,5,6,7,9,10,11);
```

```
        .data
block:  .byte 0,0,0,3,3,3,6,6,6          @ block position
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;
        i = 0;
        q = g + 81;
        do { *p++ = i;
        } while p < q;

};
```

```
clear:  mov r0, r5              @ r5 has &grid [ ][ ]
        mov r1, # 0
        add r2, r0, # 81
loop:   strb r1, [r0], # 1
        cmp r0, r2
        blt loop
        mov pc, lr
```

```
int check (int i, j, d) {
        int * p, q;
        int t, k;



        conflict = 0;
        t = i * 9;
        p = g + t;
        t = chkr (p, d);

        conflict += t;
        p = g + j;

        t = chkc (p, d);

        conflict += t;
        q = &block [0];
        t = *(q + i);

        t = t * 9
        p = g + t;
        k = *(q + j)

        p = p + k;
        t = chkb (p, d);

        conflict += t;
        return (conflict);
};
```

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

```
void update (int i, j, d) {
        int * p, q;
        int t;
        t = i * 9;
        p = g + t;
        p = p + j;
        * p = d;
        q = &place [0]
        t = *(q + i);
        i = *(q + j);
        j = t;
        dispch (j, i, d+'0'));


};
```

```
update: str lr, [sp, # -4]!     @ i, j, d : r0, r1, r2
        str r4, [sp, # -4]!     @ callee saved

        add r4, r0, r0, LSL # 3 @ t : r4
        add r4, r5, r4          @ r5 has &grid [ ][ ]
        add r4, r4, r1
        strb r2, [r4]
        ldr r3, = place         @ q : r3
        ldrb r4, [r3, r0]       @ t : r4
        ldrb r0, [r3, r1]
        mov r1, r4
        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

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

int init ( ) {
int i, j, d;
 	
int n = 0;
clear ( );
dispg ( );
h = open_file (filename);
 	
do { i = readf (h, 8);
 	
j = readf (h, 8);
 	
d = readf (h, 9);
 	
if (d == 0) exit;
 	
t = check (i, j, d);
 	
if (t == 0) {
 	
update (i, j, d);
 	
n++;
};
};
close_file (h);
 	
return (n);
};
	init: str lr, [sp, # -4]!
	str r6, [sp, # -4]! @ callee saved
	str r7, [sp, # -4]! @ callee saved
	mov r6, # 0 @ n : r6
	bl clear @ clear the grid
	bl dispg @ display blank grid
	mov r1, # 0
	ldr r0, = file
	swi 0x66 @ open file
	mov r7, r0 @ h : r7
	L: mov r0, r7 @ read i (range 0..8)
	mov r1, # 8
	bl readf
	str r0, [sp, # -4]! @ push i
	mov r0, r7 @ read j (range 0..8)
	mov r1, # 8
	bl readf
	str r0, [sp, # -4]! @ push j
	mov r0, r7 @ read d (range 0..9)
	mov r1, # 9
	bl readf
	str r0, [sp, # -4]! @ push d
	cmp r0, # 0
	addeq sp, sp, # 12 @ adjust stack
	beq exit
	mov r2, r0
	ldr r1, [sp, # 4] @ j from stack
	ldr r0, [sp, # 8] @ i from stack
	bl check
	cmp r0, # 0
	addne sp, sp, # 12 @ adjust stack
	bne L
	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
	add r6, r6, # 1
	b L
	
	exit: mov r0, r7
	swi 0x68 @ close file
	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;
int grid [9] [9];
char * m1 = “Success”;
char * m2 = “Failure”;
char * m3 = “enter row, column and digit”;
```

```
        .data
grid:   .space 81
m1:     .asciz "Success"
m2:     .asciz "Failure"
m3:     .asciz "enter row column and digit\n"
        .text
        .global _start
        .extern init, get, check, update
```

```
int main ( ) { int n, i, j, d;
        g = & grid [0][0];
        n = init ( );

        do {    if (n == 81) {

                        message (m1);

                        exit;
                };
                message (m3);

                i = get (8);


                j = get (8);


                d = get (9);


                if (d == 0) {


                        message (m2);

                        exit;
                };
                t = check (i, j, d);



                if (t == 0) {


                        update (i, j, d);



                        n++;
                };
        };
};
```

```
_start:
        ldr r5, = grid          @ used as global
        bl init                 @ initialize
        mov r6, r0              @ n : r6
L:      cmp r6, # 81
        bne cont1
        ldr r0, = m1
        swi 0x02                @ success message
        b exit
cont1:
        ldr r0, = m3
        swi 0x02                @ prompt for input
        mov r0, # 8
        bl get                  @ get i (range 0..8)
        str r0, [sp, # -4]!     @ push i
        mov r0, # 8
        bl get                  @ get j (range 0..8)
        str r0, [sp, # -4]!     @ push j
        mov r0, # 9
        bl get                  @ get d (range 0..9)
        str r0, [sp, # -4]!     @ push d
        cmp r0, # 0
        bne cont2
        add sp, sp, # 12        @ adjust stack
        ldr r0, = m2
        swi 0x02
        b exit

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
        cmp r0, # 0
        addne sp, sp, # 12      @ adjust stack
        bne L
        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
        add r6, r6, # 1
        b L
exit:   swi 0x11
```

```
        .end
```