ISA PROJECT: Yotam Aharony 308338169, Lior Solomon 205893704

Assembler

The assembler contains 4 files: assembler.c, assembler.h, our_list.c, our_list.h

Assembler.c – Contains the assembler main function, receives as arguments the file pathes given by the command line (*.asm, memin.txt). Activates the assembler. It uses some utility funcations as mentiond below, and use the our_list class.

We'll read the assembler file twice:

1. Get a list of lables and their line numbers. Get the maximal address of writing (max of number of commands, and the biggest address of .word command)
2. Translate commands line by line. We'll keep it in an array (of Strings – 8 Chars)
3. At the – we'll write the array into the output file

Documentation:

```
/*
* This function goes over the input file line by line and create a List of labels and
their line number.
* It also return the max line - max(number of commands, max address of .word command)
* @param filename - the path of the input file
* @param max_line - a pointer to the integer which keeps the max_line
* @return List* - the list of the labels and their line number
*/
List* get_labels_lines(char* filename, int* max_line);

/*
* This function goes over the input file line by line and translate the assembler command
into hexa string.
* it writes it into an array. At the end, the array will be written into the output file
* It uses get_labels_lines and write_to_file methods.
* @param filename - the path of the input file
* @param output - the path of the output file
* @return - void
*/
void readFile(char* filename, char* output);

/*
* This function translates the opcode in the assembler: from opcode to its hexa
representation. For example: add->0
* @param opcode - the opcode char
* @return - the char it represents ( add = 0, sub = 1...)
*/
char opcode_to_bin(char* opcode);
```

```c
/*
* This function translates the register name in the assembler: from its name to its hexa
representation. For example: $zero->0
* @param name - the register name
* @return - the char it represents in hexa( $zero = 0, $at = 1...)
*/
char registerName_to_number(char* name);




/* translate the imm string (3 chars) into int (take care of hexa and dec). For example
0x002->, 256->100, 0x256->256…
* @param str - the imm string
* @return int - the imm as int
*/
int imm_to_int(char* str);

/*
* This function gets a file path and array of strings (and its size). It prints line by
line the array into the file.
* @param path - file path to write to
* @param arr - the array to write
* @param arr_size - the array size
* @return - void
*/
void write_to_file(char* path, char** arr, int arr_size);

/*
* This function free the array's memory
* @param arr - the strings array
* @param len - the array's length
* @return - void
*/
void destroyArray(char** arr, int len);

/*
* This main function which runs the assembler program
*/
int main(int argc, char** argv);
```

Our_list.c, our_list. h - an auxiliary list class.

```
/* A List’s node struct. Contains labelName, lineNumber and next */
 typedef struct Node {
       unsigned int line_number;
       char* label_name;
       struct Node* next; /* Pointer to next node in the list */
} Node;

/* a linked list struct. Contains the head of the list and its size */
typedef struct List {
       Node* head; /* this pointer points on the head of the list */
       size_t size;
} List;

/*
* initList
*
*  This function initializes new list structure, with a single dummy node (has a -1 value
for
*  number of moves. this dummy node will be useful for detecting if we got to the
beginning
*  of the list.
*  @return - pointer to the new list
*/
List* initList();

/**
* This function creates a new node with a given key and value.
* @param key - The key to be given to the new node
* @param value - The value to be given to the new node
* @return - A pointer to the new node created
*/
Node* create_node(char* label_name, unsigned int line_number);

/**
* This function iterates over the linked list to find the value of a node given a key
* @param list - The list to find the node in
* @param key - The key of the node to look for
* @return value - If a node with the given key is found on the list
* -1 - If list = NULL or if the list doesn't contain a node with the given key
*/
int find_node(List* list, char* label_name);

/*
* addMove
*
*  The function adds a new node (which was already prepared) to the list
*  @param undoList - pointer to the current list
*  @param newNode - pointer to a new node (which was already created and filled)
*  @return -
*/
void addNode(List* list, char* label_name, unsigned int line_number);

/*
* destroyList
*
```

```c
*  This function clears the entire list from memory.
*  @param undoList - pointer to the current list
*  @return -
*/
void destroyList(List* list);
```

Simulator

The simulator contains 4 files: Simulator.c, Simulator.h, Commands.c, Commands.h

Simulator.c – Contains the simulator main function, receives as arguments the file pathes given by the command line (**memin.txt memout.txt regout.txt trace.txt count.txt**).

The simulator reads the current memory state from memin.txt file and then simulate commands followed by the PC pointer. At start the PC pointer points to 0.

During the simulation, the trace.txt file is written, describing the change in the registers after each command was executed.
Finally, the number of command was executed, the final registers values and the final memory state written to count.txt, regout.txt and memout.txt respectively.

Documentation:

Simulator.c , Simulator.h

```
/**
 * A type to represent all opcodes of the SIMP commands
 */
typedef enum {
       ADD = 0,
       SUB = 1,
       AND = 2,
       OR = 3,
       SLL = 4,
       SRA = 5,
       MAC = 6,
       BRANCH = 7,
       JAL = 11,
       LW = 12,
       SW = 13,
       JR = 14,
       HALT = 15
}OP_CODE;

/*
* This function write data to a file by a given filepath.
* @param path - the path of the input file
* @param data - the data to write
* @param isArr - 1 in case the data is an array , 0 in case the data is a number
* @return int - on succses = 0 , on failuer = -1
*/
int writeToFile(char * path, int * data, int len, int isArr);

/*
* This function go over the int array for serching the index of the last non zero member
in a int array
* @param data - the int array
* @param len - the length of the int array
* @return int - the index of the last non zero member in a int array
*/
int getLastIndex(int * data, int len);

/*
* This function simulates the assembly program by the given memory snapshot.
* @param data - the int array
* @param len - the length of the int array
* @return int - the index of the last non zero member in a int array
*/
int simulate(int * mem, int * regs, char * trace_path);

/**
 * This function converts a hexadecimal number from a string to a decimal integer
 * @param hexVal - The string containing the hexadecimal number
 * @return int - The decimal value of hex_str
 */
int hex2int(char hexVal[]);

/**
 * This function sets the memory snapshot by reading from memin file.
 * @param fileName - The path of memin.txt file
 * @param mem - A pointer to the memory array
 * @return 0 - On success, if the command is not HALT
 * -1 - If the command was HALT
 */
int setMem(char *fileName, int * mem);

/**
 * This function executes a SIMP command , update the PC pointer , registers values and
memory state.
 * @param cm - The command struct contains all needed details.
 * @param mem - The current memory
 * @param regs - The current registers
 * @param pc - The current pc
 * @return 0 - On success, if the command is not HALT
 * -1 - If the command was HALT
 */
int execCommand(Command cm, int * mem, int * regs, int * pc);
```

Commands.c , Commands.h

```
/*
*       A struct that represents a SIMP processor command.
*/
typedef struct command
{
        char opcode;
        int rd;
        int rs;
        int rt;
        int rm;
        int imm;
}Command;
```

```c
/* add operation */
void do_add(int * regs, int rd, int rs, int rt, int imm);

/* sub operation */
void do_sub(int * regs, int rd, int rs, int rt, int imm);

/* and operation */
void do_and(int * regs, int rd, int rs, int rt, int imm);

/* or operation */
void do_or(int * regs, int rd, int rs, int rt, int imm);

/* sll operation */
void do_sll(int * regs, int rd, int rs, int rt, int imm);

/* sra operation */
void do_sra(int * regs, int rd, int rs, int rt, int imm);

/* mac operation */
void do_mac(int * regs, int rd, int rs, int rt, int rm, int imm);

/* branch operation */
void do_branch(int regs[], int * pc, int rd, int rs, int rt, int rm, int imm);

/* jal operation */
void do_jal(int * regs,int * pc, int imm);

/* lw operation */
void do_lw(int *regs, int mem[], int rd, int rs, int rt, int imm);

/* sw operation */
void do_sw(int regs[], int * mem, int rd, int rs, int rt, int imm);

/* jr operation */
void do_jr(int regs[], int * pc, int rd);
```