מבנה המחשב - פרויקט - תיעוד

207232224 318912961,

6 ביוני 2022

1 אסמבלר

1.1 עיקרון פעולה

האסמבלר מקבל כקלט קובץ אסמבלי. על קובץ זה האסמבלר עובר פעמיים, בפעם הראשונה הוא עובר על כלל הלייבלים ומחליף אותם במספר שאותו הם מייצגים, ובמעבר השני מתרגם את קוד האסמבלי לאופקוד המייצג אותו.

1.2 dict

קובץ זה הוא קובץ המשמש כמילון המתרגם סטרינג שבו השם של הרגיסטר או הפקודה, למספר המייצג את המספר של הרגיסטר או הפקודה שהוכנסו.

הגדרות הרגיסטרים:

```
#define zero "zero"
#define imm "imm"
#define v0 "v0"
#define a0 "a0"
#define a1 "a1"
#define a2 "a2"
#define a3 "a3"
#define t0 "t0"
#define t1 "t1"
#define t2 "t2"
#define s0 "s0"
#define s1 "s1"
#define s2 "s2"
#define gp "gp"
#define sp "sp"
#define ra "ra"
```

הגדרות הפקודות:

```
#define word ".word"
#define add "add"
#define sub "sub"
#define mul "mul"
#define and "and"
#define or "or"
#define xor "xor"
#define sll "sll"
#define sra "sra"
#define srl "srl"
#define beq "beq"
#define bne "bne"
#define blt "blt"
#define bgt "bgt"
#define ble "ble"
#define bge "bge"
#define jal "jal"
#define lw "lw"
#define sw "sw"
#define reti "reti"
#define in "in"
#define out "out"
#define halt "halt"
```

השוואה בין הביטויים:

```
int compare(char *exp) // returns the expression's opcode
{
// * Registers
if (!strcmp(exp, zero)) return 0;
else if (!strcmp(exp, imm)) return 1;
else if (!strcmp(exp, v0)) return 2;
else if (!strcmp(exp, a0)) return 3;
else if (!strcmp(exp, a1)) return 4;
else if (!strcmp(exp, a2)) return 5;
else if (!strcmp(exp, a3)) return 6;
else if (!strcmp(exp, t0)) return 7;
else if (!strcmp(exp, t1)) return 8;
else if (!strcmp(exp, t2)) return 9;
else if (!strcmp(exp, s0)) return 10;
else if (!strcmp(exp, s1)) return 11;
else if (!strcmp(exp, s2)) return 12;
else if (!strcmp(exp, gp)) return 13;
else if (!strcmp(exp, sp)) return 14;
else if (!strcmp(exp, ra)) return 15;
// * OpCodes
else if (!strcmp(exp, word)) return -1;
else if (!strcmp(exp, add)) return 0;
else if (!strcmp(exp, sub)) return 1;
else if (!strcmp(exp, mul)) return 2;
else if (!strcmp(exp, and)) return 3;
else if (!strcmp(exp, or)) return 4;
else if (!strcmp(exp, xor)) return 5;
else if (!strcmp(exp, sll)) return 6;
else if (!strcmp(exp, sra)) return 7;
else if (!strcmp(exp, srl)) return 8;
else if (!strcmp(exp, beq)) return 9;
else if (!strcmp(exp, bne)) return 10;
else if (!strcmp(exp, blt)) return 11;
else if (!strcmp(exp, bgt)) return 12;
else if (!strcmp(exp, ble)) return 13;
else if (!strcmp(exp, bge)) return 14;
else if (!strcmp(exp, jal)) return 15;
else if (!strcmp(exp, lw)) return 16;
else if (!strcmp(exp, sw)) return 17;
else if (!strcmp(exp, reti)) return 18;
else if (!strcmp(exp, in)) return 19;
else if (!strcmp(exp, out)) return 20;
else if (!strcmp(exp, halt)) return 21;
}
```

1.3 label

קובץ זה מגדיר מבנה נתונים חדש בשם לייבל.

הגדרת לייבל חדש:

```
Label *labelNewLabel(char name[], int location)
{
Label *new_label = (Label *)malloc(sizeof(Label));
if (new_label != NULL) // making sure the memory allocation succeed
{
strcpy(new_label->name, name);
new_label->next = NULL;
new_label->location = location;
}
return new_label;
}
```

מציאת הלייבל האחרון:

```
Label *labelLast(Label *head)
{
Label *temp = head;
while (temp->next != NULL)
{
temp = temp->next;
}
return temp;
}
```

הוספת המידע על הלייבל החדש:

```
void labelAppendNode(Label *head, Label *node)
{
Label *tail = labelLast(head); // get the last node
tail->next = node;
}
void labelAppendData(Label *head, char name[], int location)
{
Label *new_tail = labelNewLabel(name, location); // creatin a new node
labelAppendNode(head, new_tail); // appending it to the end of the list
}
```

מציאת הלייבל בעזרת האינדקס:

```
Label *labelGetByIndex(Label *head, int index)
{
if (index < 0) // making sure the index is valid
{
return NULL;
}
Label *temp = head;
int i;
for (i = 0; i < index; i++)
{
if (temp != NULL) // if we are not out of range
{
temp = temp->next;
}
else
{
break;
}
}
return temp;
}
```

מציאת הלייבל בעזרת שמו:

```
Label *labelGetByName(Label *head, char name[])
{
Label *temp = head;
int i;
while (strcmp(temp->name, name))
{
temp = temp->next;
if (temp == NULL) // while we are no out of range
{
break;
}
}
return temp;
}
```

בדיקה האם הלייבל קיים:

```
int labelListContains(Label *head, char name[])
{
while (head != NULL)
{
if (strcmp(head->name, name) == 0)
{
return TRUE;
}
head = head->next;
}
return FALSE;
}
```

מחיקת הלייבלים בסוף ריצת הקוד:

```
void labelDeleteList(Label *head)
{
Label *temp; // A pointer to a Label we are going to destroy after updating head
while (head != NULL) // Destroy all of the list
{
temp = head; // temp <-- current node, head <-- head->next
head = head->next;
free(temp); // we destroy temp and free the memory
}
}
```

הדפסת כל הלייבלים:

```
void labelPrintList(Label *head)
{
while (head != NULL)
{
printf("Name: %s\n Location:%d\n", head->name, head->location);
head = head->next;
}
}
```

1.4 Main

קובץ זה מכיל את הפונקציה הראשית של האסמבלר ואת כל מהלכו.
ריצת האסמבלר מתחלקת לשני מעברים על הקובץ:

1. מציאת ושמירת כלל הלייבלים
2. תרגום האופקודים, הרגיסטרים והלייבלים, ושמירתם בסטרינג ארוך שמייצג את memin

לאחר שני המעברים הללו הסטרינג נכתב בקובץ memin.

הגדרות:

```
# define MAX_LINE_SIZE 501
# define GET_LABEL 0
# define TRANSLATE_ITER 1
# define MAX_MEMIN_SIZE 4096
# define TRUE 1
# define FALSE 0
int memin_loc = 0;
```

בדיקה האם ההוראה היא I-Type או R-Type:

```
int is_imm(char *line) // checks if the line has an imm (not in the 1st reg)
{
int counter = 0;
int jump = 0;
for (int i = 0; i < MAX_LINE_SIZE; i++) // for every char in the line
{
if (i == 0)
{
if (isalpha(line[i])) counter++;
}
else if (line[i] == '$' && isspace(line[i-1])) counter++;
if (line[i] == 'b' && line[i+1] == 'e' && line[i+2] == 'q') jump=1; // if the op is beq
if (line[i] == 'b' && line[i+1] == 'n' && line[i+2] == 'e') jump=1; // if the op is bne
if (line[i] == 'b' && line[i+1] == 'l' && line[i+2] == 't') jump=1; // if the op is blt
if (line[i] == 'b' && line[i+1] == 'g' && line[i+2] == 't') jump=1; // if the op is bgt
if (line[i] == 'b' && line[i+1] == 'l' && line[i+2] == 'e') jump=1; // if the op is ble
if (line[i] == 'b' && line[i+1] == 'g' && line[i+2] == 'e') jump=1; // if the op is bge
if (line[i] == 's' && line[i+1] == 'w') jump=1; // if the op is sw
if (line[i] == 'o' && line[i+1] == 'u' && line[i+2] == 't') jump=1; // if the op is out
if (line[i] == 'i' && line[i+1] == 'm' && line[i+2] == 'm' && counter != 2) return TRUE;
if (line[i] == 'i' && line[i+1] == 'm' && line[i+2] == 'm' && counter == 2 && jump == 1)
return TRUE;
// if (line[i] == 'i' && line[i+1] == 'm' && line[i+2] == 'm' && counter == 1) return FALSE;
}
return FALSE;
}
```

הוספת לייבל חדש:

```
void add_label(char *line, int line_loc, Label *label_list) // add new label to the label
list
{
char label [MAX_LINE_SIZE] = "";
int counter = 0;
for (int i = 0; line[i] != ':'; i++) // iterate until ":"
{
if (isspace(line[i])) counter++;
else label[i-counter] = line[i];
}
labelAppendData(label_list, label, line_loc); // add label to list
}
```

חיפוש לייבל:

```
int search_label(char *line, int line_index, int line_loc, Label *label_list) // iterate a
line and check if there is a label in it,
{
for (int i = 0; i < MAX_LINE_SIZE; i++)
{
if (line[i] == '\0' || line[i] == '\n') break;
if (line[i] == ':')
{
add_label(line, line_loc, label_list); // if the line has ":"  add the label to the list
with it's location
return --line_loc;
}
}
if (is_imm(line)) line_loc++;
return line_loc;
}
```

הוספת המידע ל-memin:

```
void add_to_memin_str(char *temp_str, char *memin_str, int size, int place) // add string
to the memin string
{
if (size == 5 || size == 2)
{
for (int i=0; i<size; i++)
{
memin_str[memin_loc*6+i] = temp_str[i]; // add the string to it's place
}
if (size == 5) memin_loc++;
}
else if (size == 1)
{
memin_str[memin_loc*6+place+1] = temp_str[0]; // add the string to it's place
if (place == 3) memin_loc++;
}
}
```

ריפוד באפסים \ באחדות:

```
int extend_sign(int num) // extend sign of the number
{
int mask_extend = 0xFFF00000; //sign externtion
int mask_msb = 0x80000; // mast to determine the msb
int sign = mask_msb & num;
//if the msb is not 0
if (sign > 0) {
num = mask_extend | num;
}
return num;
}
```

הוספת המידע של הפקודה word.:

```
void add_word(char *line, char *memin_str) // add .word data to memin
{
int line_loc = 0;
char line_val[MAX_LINE_SIZE];
char temp_var[MAX_LINE_SIZE];
int counter = 0;
int j = 0;
int hex = 0;
for (int i = 0; i < MAX_LINE_SIZE; i++)
{
if (line[i] == '.') // skip the "word" and move to the rest of the line
{
i += 4;
continue;
}
if (isalnum(line[i]))
{
if (line[i] == '0' && (line[i+1] == 'x' || line[i+1] == 'X')) // if the number is hex
hex = 1;
temp_var[j] = line[i]; // save the var's chars in string
if (isspace(line[i+1]) || (line[i+1] == '\0')) // when the number ends
{
if (counter == 0) // line place
{
if (hex == 1) line_loc = extend_sign(strtoul(temp_var, NULL, 16));
else line_loc = atoi(temp_var);
j = 0;
counter++;
hex = 0;
continue;
}
else if (counter == 1) // line data
{
temp_var[j+1] = '\0';
if (hex == 1)
{
for (int k = 2; k < 7; k++) line_val[k - 2] = temp_var[k]; // remove "0x" from data
int len = strlen(line_val);
if (len < 5) // zero padding
{
for (int l=0; l<5-len; l++)
{
for (int k=0; k<5; k++)
{
line_val[5-k] = line_val[5-k-1];
}
}
for (int k=0; k<5-len; k++)
{
line_val[k] = '0';
}
}
}
else sprintf(line_val, "%05X", atoi(temp_var));
break;
}
}
j++;
}
if ((line[i] == '#') || (line[i] == '\n') || (line[i] == '\0')) break;
if (isspace(line[i])) continue;
}
int memin_loc_temp = memin_loc;
memin_loc = line_loc;
add_to_memin_str(line_val, memin_str, 5, 0);
memin_loc = memin_loc_temp;
}
```

תרגום קוד האסמבלי:

```
void translate_file(char *line, int line_index, int line_loc, Label *label_list, FILE *memin, char *memin_str) // switch asm with
it's opcode
{
char var[MAX_LINE_SIZE] = "";
int counter = 0;
int len = 0;
int hex;
int hex_num = 0;
Label *label;
char temp_str[7];
char temp_char[2];
int not_imm = 0;
for (int i = 0; i < strlen(line); i++) // check if the imm is hex
{
if ((line[i] == '0') && (line[i+1] == 'x' || line[i+1] == 'X'))
{
hex_num = 1;
break;
}
}
for (int i = 0; i < strlen(line); i++)
{
if (line[i] == ':') return; // skip label line
else if (counter == 4 && isalpha(line[i]) && hex_num != 1) // translate label
{
for (int j=i; !isspace(line[j]); j++)
{
var[len] = line[j];
var[++len] = '\0';
}
label = labelGetByName(label_list, var);
sprintf(temp_str, "%05X", label -> location);
add_to_memin_str(temp_str, memin_str, 5, 0);
return;
}
else if (line[i] == ',' || line[i] == '$' || line[i] == '#' || line[i] == '\n' || i == strlen(line)-1) // translate name
{
if (i == strlen(line)-1) // if the line is finished close the var
{
var[len] = line[i];
var[++len] = '\0';
}
if (var[0] == '0' && (var[1] == 'x' || var[1] == 'X') && not_imm == 0) // translate hex imm num
{
strcpy(temp_str, var+sizeof(char)*2);
int number = (int)strtoul(temp_str, NULL, 16);
sprintf(temp_str, "%05X", number&0x000FFFFF);
add_to_memin_str(temp_str, memin_str, 5, 0);
return;
}
else if ((isdigit(var[0]) || (var[0] == '-')) && atoi(var) != 0 && not_imm == 0) // translate dec imm num
{
sprintf(temp_str, "%05X", atoi(var)&0x000FFFFF);
add_to_memin_str(temp_str, memin_str, 5, 0);
return;
}
else if (is_imm(line) && atoi(var) == 0 && counter == 4 && not_imm == 0) // translate 0 imm num
{
sprintf(temp_str, "%05X", atoi(var));
add_to_memin_str(temp_str, memin_str, 5, 0);
return;
}
else if (strlen(var) != 0 && !isdigit(var[0])) // translate opcodes and regs
{
hex = compare(var);
if (hex == -1) // if the opcode is ".word"
{
add_word(line, memin_str);
break;
}
else
{
if (counter == 0) // translate opcode
{
sprintf(temp_str, "%02X", hex);
add_to_memin_str(temp_str, memin_str, 2, 0);
}
else // translate reg
{
if ((hex == 1) && (counter == 1)) not_imm = 1;
if ((hex == 1) && (counter != 1)) not_imm = 0;
sprintf(temp_char, "%01X", hex);
add_to_memin_str(temp_char, memin_str, 1, counter);
}
}
var[0] = '\0';
len = 0;
counter++;
}
if ((line[i] == '#') || (line[i] == '\n')) break;
}
else if (isspace(line[i])) continue;
else
{
var[len] = line[i];
var[++len] = '\0';
}
}
return;
}
```

מעבר על כלל השורות:

```
void iter_lines(FILE *fp, char iter_type, Label *label_list, FILE *memin, char *memin_str)
// iterate over all the lines in the file
{
FILE *asm_file = fp;
char line [MAX_LINE_SIZE];
int line_index = 0;
int line_loc = 0;
int is_empty = 0;
while (fgets(line, MAX_LINE_SIZE, asm_file))
{
for (int i = 0; i < strlen(line); i++) // check if line is empty
{
if (!(line[i] == ' '))
{
if (line[i] != '\n' && line[i] != '#') break;
else
{
is_empty = 1;
break;
}
}
}
if (is_empty == 1)
{
is_empty = 0;
continue;
}
if (iter_type == GET_LABEL) line_loc = search_label(line, line_index, line_loc, label_list);
// iterate and find all labels
else if (iter_type == TRANSLATE_ITER) translate_file(line, line_index, line_loc, label_list,
memin, memin_str); // iterate and translate to hex
line_index++;
line_loc++;
}
}
```

כתיבת המידע לקובץ המוצא בסוף הריצה:

```
void write_to_file(char *memin_str, FILE *memin) // write output to memin file
{
for (int i=0; i<MAX_MEMIN_SIZE*6-1; i++) fputc(memin_str[i], memin);
}
```

פונקציית ה-main:

```
int main(int arg_amount, char *arg_vals[])
{
FILE *asm_file = fopen(arg_vals[1], "r");
if (asm_file)
{
char memin_str[MAX_MEMIN_SIZE*6+1];
for (int i=0; i<MAX_MEMIN_SIZE*6; i++) // zero the memin_str, and put {Enter} after 5 chars
{
if (((i+1)%6) == 0) memin_str[i] = '\n';
else memin_str[i] = '0';
}
memin_str[0] = '0';
memin_str[MAX_MEMIN_SIZE*6] = '\0';
FILE *memin = fopen(arg_vals[2], "w");
Label *label_list = labelNewLabel("", -1);
iter_lines(asm_file, GET_LABEL, label_list, memin, memin_str); // iterate and find all labels
fseek(asm_file, 0, 0); // go to the beginning of the file
iter_lines(asm_file, TRANSLATE_ITER, label_list, memin, memin_str); // iterate and translate
to hex
write_to_file(memin_str, memin); // write output to memin file
fclose(memin);
labelDeleteList(label_list); // delete all of the labels
fclose(asm_file);
}
return 0;
}
```

2 סימולטור

פעולת הסימולטור בוצעה באופן הבא: כלל קבצי הכניסה נכתבו למערכים. עבור כל איטרציה נלקחה שורה מקובץ ה-memin ותורגמה לפעולה שיש לבצע ולרגיסטרים שיש לבצע אותה עליהם, ולאחר מכן בוצעה הפעולה בהתחשב בתנאים הנתונים (כמו למשל טיימר או כתיבה לדיסק).

2.1 dict

גם בסימולטור השתמשנו בקובץ המשמש מילון המתרגם את המספר לרגיסטרים שיש בהם שימוש.
השוואת הרגיסטרים:

```
if (!strcmp(exp, zero))
return 0;
else if (!strcmp(exp, imm_dict))
return 1;
else if (!strcmp(exp, v0))
return 2;
else if (!strcmp(exp, a0))
return 3;
else if (!strcmp(exp, a1))
return 4;
else if (!strcmp(exp, a2))
return 5;
else if (!strcmp(exp, a3))
return 6;
else if (!strcmp(exp, t0))
return 7;
else if (!strcmp(exp, t1))
return 8;
else if (!strcmp(exp, t2))
return 9;
else if (!strcmp(exp, s0))
return 10;
else if (!strcmp(exp, s1))
return 11;
else if (!strcmp(exp, s2))
return 12;
else if (!strcmp(exp, gp))
return 13;
else if (!strcmp(exp, sp))
return 14;
else if (!strcmp(exp, ra))
return 15;
```

השוואת הפעולות:

```
else if (!strcmp(exp, word))
return -1;
else if (!strcmp(exp, add))
return 0;
else if (!strcmp(exp, sub))
return 1;
else if (!strcmp(exp, mul))
return 2;
else if (!strcmp(exp, and))
return 3;
else if (!strcmp(exp, or ))
return 4;
else if (!strcmp(exp, xor))
return 5;
else if (!strcmp(exp, sll))
return 6;
else if (!strcmp(exp, sra))
return 7;
else if (!strcmp(exp, srl))
return 8;
else if (!strcmp(exp, beq))
return 9;
else if (!strcmp(exp, bne))
return 10;
else if (!strcmp(exp, blt))
return 11;
else if (!strcmp(exp, bgt))
return 12;
else if (!strcmp(exp, ble))
return 13;
else if (!strcmp(exp, bge))
return 14;
else if (!strcmp(exp, jal))
return 15;
else if (!strcmp(exp, lw))
return 16;
else if (!strcmp(exp, sw))
return 17;
else if (!strcmp(exp, reti))
return 18;
else if (!strcmp(exp, in))
return 19;
else if (!strcmp(exp, out))
return 20;
else if (!strcmp(exp, halt))
return 21;
```

2.2 instruction

קובץ זה ממש פיצול הקוד של הפקודה לפעולה, הרגיסטרים ומיקומה של הפעולה. כאשר כלל הפרמטרים נשמרים במבנה חדש בשם Instruction.

יצירת הוראה חדשה:

```
Instruction* instructionNewinstruction(int opcode, int rd, int rs, int rt, int imm, int
location)
{
Instruction* new_instruction = (Instruction*)malloc(sizeof(Instruction));
if (new_instruction != NULL)
{
new_instruction->rt = rt;
new_instruction->rs = rs;
new_instruction->rd = rd;
new_instruction->opcode = opcode;
new_instruction->imm = imm;
new_instruction->location = location;
new_instruction->next = NULL;
}
return new_instruction;
}
```

מציאת ההוראה על ידי מיקומה:

```
Instruction* instructionGetByLocation(Instruction* head, int location)
{
Instruction* temp = head; //temporary node repersenting the current node checked.
while (temp->location != location) // while the current node is not the one we are searching
for
{
temp = temp->next; //current node would set to the next node.
if (temp == NULL) //if we got to the last node of the lisr -> exit and return null
{
break;
}
}
return temp;
}
```

מציאת ההוראה האחרונה:

```
Instruction* instructionGetByLocation(Instruction* head, int location)
{
Instruction* temp = head; //temporary node repersenting the current node checked.
while (temp->location != location) // while the current node is not the one we are searching
for
{
temp = temp->next; //current node would set to the next node.
if (temp == NULL) //if we got to the last node of the lisr -> exit and return null
{
break;
}
}
return temp;
}
```

הוספת מיקום להוראה נוספת:

```
void instructionAppendNode(Instruction* head, Instruction* node)
{
Instruction* tail = instructionLast(head); //get the last node in the list
tail->next = node; //append the node
}
```

פענוח סוג ההוראה (I-Type \ R-Type) ממבנה ההוראה:

```
int instructionType(Instruction* inst)
{
if ((inst->rd == IMM_REG) && ((beq_opcode <= inst->opcode && inst->opcode <= bge_opcode) ||
inst->opcode==sw_opcode || inst->opcode == out_opcode))
{
return I_TYPE;
}
if ((inst->rs == IMM_REG) || (inst->rt == IMM_REG)) {
return I_TYPE;
}
return R_TYPE;
}
```

פענוח סוג ההוראה משורת ההוראה:

```
int instructionTypeFromLine(char* line)
{
//if one of the registers in the line is the immidiate register
int rt = slice_atoi_hex(line, 4, 5);//set rt to its numeric value
int rs = slice_atoi_hex(line, 3, 4);//set rs to its numeric value
int rd = slice_atoi_hex(line, 2, 3);//set rd to its numeric value
int opcode = slice_atoi_hex(line, 0, 2);//set opcode to its numeric value
if ((rd == IMM_REG) && ((beq_opcode <= opcode && opcode <= bge_opcode) || opcode == sw_opcode
|| opcode == out_opcode)) {
return I_TYPE;
}
if ((rs == IMM_REG) || (rt == IMM_REG)) {
return I_TYPE;
}
return R_TYPE;
}
```

הוספת מידע להוראה חדשה:

```
void instructionAppendData(Instruction* head, int opcode, int rd, int rs, int rt, int imm,
int location)
{
//create a new instruction from the given parameters
Instruction* new_tail = instructionNewinstruction(opcode, rd, rs, rt, imm, location);
//append the new node
instructionAppendNode(head, new_tail);
}
```

הוספת הוראה חדשה משורה:

```
void instructionAppendFromLine(Instruction* head, char* line, char* imm_line, int location)
{
int rt = slice_atoi_hex(line, 4, 5); //set rt to to its numeric value
int rs = slice_atoi_hex(line, 3, 4); //set rs to to its numeric value
int rd = slice_atoi_hex(line, 2, 3); //set rd to to its numeric value
int opcode = slice_atoi_hex(line, 0, 2); //set opcode to to its numeric value
int imm = 0;
if (instructionTypeFromLine(line) == I_TYPE) // if the instruction is of I type
{
imm = extend_sign(strtoul(imm_line, NULL, 16)); //set imm to to its numeric value
}
instructionAppendData(head, opcode, rd, rs, rt, imm, location); //append the list with the
new node
}
```

פענוח ההוראה מהשורה:

```
Instruction* instructionFromLine(char* line, char* imm_line, int location)
{
int rt = slice_atoi_hex(line, 4, 5);//set rt to to its numeric value
int rs = slice_atoi_hex(line, 3, 4);//set rs to to its numeric value
int rd = slice_atoi_hex(line, 2, 3);//set rd to to its numeric value
int opcode = slice_atoi_hex(line, 0, 2);//set opcode to to its numeric value
int imm = 0;
if (instructionTypeFromLine(line) == I_TYPE)// if the instruction is of I type
{
imm = extend_sign(strtoul(imm_line, NULL, 16));//set imm to to its numeric value
}
// create a new Instruction node with the given parameters taken from the line
Instruction* new_inst = instructionNewinstruction(opcode, rd, rs, rt, imm, location);
return new_inst;
}
```

פיצול סטרינג הקסהדצימלי והחזרתו כ-int:

```
int slice_atoi_hex(char str[], int start, int end)
{
int len = end - start;
char tmp[LINE_MAX_SIZE];
strncpy(tmp, str + start, len); //copy the wanted part of the line to tmp
tmp[len] = '\0'; //add null at the end of the string
return strtoul(tmp, NULL, 16); //return its numeric value
}
```

ריפוד באפסים \ באחדות:

```
int extend_sign(int num) // extend sign of the number
{
int mask_extend = 0xFFF00000; //sign externtion
int mask_msb = 0x80000; // mast to determine the msb
int sign = mask_msb & num;
//if the msb is not 0
if (sign > 0) {
num = mask_extend | num;
}
return num;
}
```

תרגום int לאופקוד:

```
char* int_to_opcode(int opcode)
{
char* opcode_srt = NULL;
opcode_srt = (char*)malloc(sizeof(char) * 20);
switch (opcode)
{
case 0: // add
strcpy(opcode_srt, add);
break;
case 1: // sub
strcpy(opcode_srt, sub);
break;
case 2: // mul
strcpy(opcode_srt, mul);
break;
case 3: // and
strcpy(opcode_srt, and);
break;
case 4: // or
strcpy(opcode_srt, or );
break;
case 5: // xor
strcpy(opcode_srt, xor);
break;
case 6: // sll
strcpy(opcode_srt, sll);
break;
case 7: // sra
strcpy(opcode_srt, sra);
break;
case 8: // srl
strcpy(opcode_srt, srl);
break;
case 9: // beq
strcpy(opcode_srt, beq);
break;
case 10: // bne
strcpy(opcode_srt, bne);
break;
case 11: // blt
strcpy(opcode_srt, blt);
break;
case 12: // bgt
strcpy(opcode_srt, bgt);
break;
case 13: // ble
strcpy(opcode_srt, ble);
break;
case 14: // bge
strcpy(opcode_srt, bge);
break;
case 15: // jal
strcpy(opcode_srt, jal);
break;
case 16: // lw
strcpy(opcode_srt, lw);
break;
case 17: // sw
strcpy(opcode_srt, sw);
break;
case 18: // reti
strcpy(opcode_srt, reti);
break;
case 19: // in
strcpy(opcode_srt, in);
break;
case 20: // out
strcpy(opcode_srt, out);
break;
case 21: // halt
strcpy(opcode_srt, halt);
break;
default:
break;
}
return opcode_srt;
}
```

תרגום int לרגיסטר:

```
char* int_to_reg(int reg)
{
char* reg_srt = (char*)malloc(sizeof(char) * 20);
switch (reg)
{
case 0:
strcpy(reg_srt, zero);
break;
case 1:
strcpy(reg_srt, imm_dict);
break;
case 2:
strcpy(reg_srt, v0);
break;
case 3:
strcpy(reg_srt, a0);
break;
case 4:
strcpy(reg_srt, a1);
break;
case 5:
strcpy(reg_srt, a2);
break;
case 6:
strcpy(reg_srt, a3);
break;
case 7:
strcpy(reg_srt, t0);
break;
case 8:
strcpy(reg_srt, t1);
break;
case 9:
strcpy(reg_srt, t2);
break;
case 10:
strcpy(reg_srt, s0);
break;
case 11:
strcpy(reg_srt, s1);
break;
case 12:
strcpy(reg_srt, s2);
break;
case 13:
strcpy(reg_srt, gp);
break;
case 14:
strcpy(reg_srt, sp);
break;
case 15:
strcpy(reg_srt, ra);
break;
default:
break;
}
return reg_srt;
}
```

הדפסת הוראה:

```
void instructionPrintInstruction(Instruction* inst)
{
char* opcode = int_to_opcode(inst->opcode);
char* rd = int_to_reg(inst->rd);
char* rs = int_to_reg(inst->rs);
char* rt = int_to_reg(inst->rt);
if (instructionType(inst) == R_TYPE)
{
printf("Location: %d DATA: %s %s %s %s\n", inst->location, opcode, rd, rs, rt);
}
else
{
printf("Location: %d DATA: %s %s %s %s Imm: %d\n", inst->location, opcode, rd, rs, rt,
inst->imm);
}
free(opcode);
free(rd);
free(rs);
free(rt);
}
```

הדפסת הוראה בהקסה:

```
void instructionPrintInstructionHex(Instruction* inst)
{
if (instructionType(inst) == R_TYPE)
{
printf("Location: %d DATA: %X %X %X %X\n", inst->location, inst->opcode, inst->rd, inst->rs,
inst->rt);
}
else
{
printf("Location: %d DATA: %X %X %X %X Imm: %X\n", inst->location, inst->opcode, inst->rd,
inst->rs, inst->rt, inst->imm);
}
}
```

מחיקת רשימת ההוראת (בסוף הריצה):

```
void instructionDeleteList(Instruction* head)
{
Instruction* temp; // A pointer to a line we are going to destroy after updating head
while (head != NULL) // Destroy all of the list
{
temp = head; // temp <-- current node, head <-- head->next
head = head->next;
free(temp); // we destroy temp and free the memory
}
}
```

2.3 IO

קובץ זה ממש את הפעולות עם הכניסות והמוצאים (Inputs \ Outputs) של הסימולטור.

מימוש הטיימר:

```
int timer(int ioreg[])
{
if (++ioreg[timercurrent] == ioreg[timermax]) // check if timer is at the max val
{
ioreg[irq0status] = 1; // activate irqstatus0
ioreg[timercurrent] = 0; // zero timer_current
}
return;
}
```

קריאת irq:

```
int irq(int ioreg[], int* pc, int *is_task)
{
ioreg[irqreturn] = *pc;
*pc = ioreg[irqhandler];
*is_task = 1;
}
```

תפעול ה-IO (פונקציית ה-main של הקובץ):

```
void IO_handler(int ioreg[], int monitor_arr[], char disk_memory[][MAX_DISK_LINE], int* pc,
int* is_task, int irq2[], int *disk_cycle, char memory[LINES_MAX][LINES_MAX_SIZE], int *led,
FILE *leds_file, FILE *display7seg_file)
{
if (ioreg[25] != *pc)
{
ioreg[25] = *pc;
if (ioreg[timerenable] == 1) // if the timer is enabled
timer(ioreg); // update processor time
}
int is_irq2 = in_irq2(ioreg,irq2);
if (*is_task != 1) // if in task
if ((ioreg[irq0enable] && ioreg[irq0status]) || (ioreg[irq1enable] && ioreg[irq1status]) ||
(ioreg[irq2enable] && (is_irq2 == 1)))
irq(ioreg, pc, is_task);
monitor(monitor_arr, ioreg);
disk_command(ioreg, disk_memory, disk_cycle, memory);
led_write(ioreg, led, leds_file, pc);
display7seg_write(display7seg_file, ioreg, pc);
}
```

הוספת תוכן קובץ irq2 למערך המתאר את האינדקסים שבהם יש לקרוא לפסיקה:

```
void IO_handler(int ioreg[], int monitor_arr[], char disk_memory[][MAX_DISK_LINE], int* pc,
int* is_task, int irq2[], int *disk_cycle, char memory[LINES_MAX][LINES_MAX_SIZE], int *led,
FILE *leds_file, FILE *display7seg_file)
{
if (ioreg[25] != *pc)
{
ioreg[25] = *pc;
if (ioreg[timerenable] == 1) // if the timer is enabled
timer(ioreg); // update processor time
}
int is_irq2 = in_irq2(ioreg,irq2);
if (*is_task != 1) // if in task
if ((ioreg[irq0enable] && ioreg[irq0status]) || (ioreg[irq1enable] && ioreg[irq1status]) ||
(ioreg[irq2enable] && (is_irq2 == 1)))
irq(ioreg, pc, is_task);
monitor(monitor_arr, ioreg);
disk_command(ioreg, disk_memory, disk_cycle, memory);
led_write(ioreg, led, leds_file, pc);
display7seg_write(display7seg_file, ioreg, pc);
}
```

בדיקה האם בפעולה הנוכחית פסיקה 2 פועלת:

```
int in_irq2(int ioreg[], int *irq2) // check if the pc should raise irq2status
{
for (int i=0; irq2[i] != -1; i++)
{
if(ioreg[clks] == *(irq2+i)) return(1);
}
return(0);
}
```

הוספת ערכים למוניטור:

```
void monitor(int monitor_arr[], int ioreg[]) // print to monitor
{
if (ioreg[monitorcmd] == 1)
{
monitor_arr[ioreg[monitoraddr]] = ioreg[monitordata];
ioreg[monitorcmd] = 0;
}
}
```

תפעול (קריאה וכתיבה) של הדיסק:

```
void disk_command(int ioreg[], char disk_memory[][MAX_DISK_LINE], int *disk_cycle, char
memory[LINES_MAX][LINES_MAX_SIZE]) // write to or read from disk
{
int is_full = 0;
if ((ioreg[diskcmd] != 0) && (*disk_cycle == 0)) // if there is a disk cmd and the disk is
available
{
if (ioreg[diskstatus] == 0) // if disk is not busy
{
*disk_cycle = 1024;
ioreg[diskstatus] = 1;
if (ioreg[diskcmd] == 1) // read sector
{
for (int i=0; i<128; i++) strcpy(memory[ioreg[diskbuffer]+i],
disk_memory[ioreg[disksector]*SECTOR_SIZE+i]); // read from disk
}
else if (ioreg[diskcmd] == 2) // write sector
{
for (int i=0; i<128; i++)
{
strcpy(disk_memory[ioreg[disksector]*SECTOR_SIZE+i], memory[ioreg[diskbuffer]+i]); // write
to disk
}
}
}
}
else if (*disk_cycle > 1)
{
*disk_cycle = *disk_cycle - 1; // if the disk is not available decrease 1 from cycles until
available
}
else if (*disk_cycle == 1) // declare the disk as available next cycle
{
ioreg[diskcmd] = 0;
ioreg[diskstatus] = 0;
ioreg[irq1status] = 1;
*disk_cycle = 0;
}
}
```

כתיבה לקובץ הלדים:

```
void led_write(int ioreg[], int *led, FILE *leds_file, int *pc)
{
if (ioreg[leds] != ioreg[23])
{
ioreg[23] = ioreg[leds];
fprintf(leds_file, "%d %08X\n", ioreg[clks], ioreg[23]); // write to leds
}
}
```

כתיבה ל-trace של רגיסטרי החומרה:

```
void hwregtrace_write(FILE *fp, int cycle, int read_write, int reg_num, int data) // write hwregtrace file
{
unsigned int data_unsigned = (unsigned) data;
char action[6] = {0};
char reg_name[50] = {0};
if (read_write == 0) sprintf(action, "READ");
else if (read_write == 1) sprintf(action, "WRITE");
switch(reg_num)
{
case irq0enable:
sprintf(reg_name, "irq0enable");
break;
case irq1enable:
sprintf(reg_name, "irq1enable");
break;
case irq2enable:
sprintf(reg_name, "irq2enable");
break;
case irq0status:
sprintf(reg_name, "irq0status");
break;
case irq1status:
sprintf(reg_name, "irq1status");
break;
case irq2status:
sprintf(reg_name, "irq2status");
break;
case irqhandler:
sprintf(reg_name, "irqhandler");
break;
case irqreturn:
sprintf(reg_name, "irqreturn");
break;
case clks:
sprintf(reg_name, "clks");
break;
case leds:
sprintf(reg_name, "leds");
break;
case display7seg:
sprintf(reg_name, "display7seg");
break;
case timerenable:
sprintf(reg_name, "timerenable");
break;
case timercurrent:
sprintf(reg_name, "timercurrent");
break;
case timermax:
sprintf(reg_name, "timermax");
break;
case diskcmd:
sprintf(reg_name, "diskcmd");
break;
case disksector:
sprintf(reg_name, "disksector");
break;
case diskbuffer:
sprintf(reg_name, "diskbuffer");
break;
case diskstatus:
sprintf(reg_name, "diskstatus");
break;
case reserved1:
sprintf(reg_name, "reserved");
break;
case reserved2:
sprintf(reg_name, "reserved");
break;
case monitoraddr:
sprintf(reg_name, "monitoraddr");
break;
case monitordata:
sprintf(reg_name, "monitordata");
break;
case monitorcmd:
sprintf(reg_name, "monitorcmd");
break;
}
fprintf(fp, "%d %s %s %08X\n", cycle-1, action, reg_name, data_unsigned); // print to file
}
```

כתיבה לתצוגת 7seg:

```
void display7seg_write(FILE *display7seg_file, int ioreg[], int *pc) // write to display7seg
file
{
if (ioreg[display7seg] != ioreg[24])
{
fprintf(display7seg_file, "%d %08X\n", ioreg[clks], ioreg[display7seg]); // print to file
ioreg[24] = ioreg[display7seg];
}
}
```

2.4 line

בקובץ זה מומשו פונקציות שתומכות בקריאת השורה ופיצולה לסטרינגים המייצגים את הרגיסטרים והפעולות שיש לבצע ברגיסטרים. הפונקציות שהוגדרו בקובץ זה הוגדרו עבור המבנה החדש: שורה.

יצירת שורה חדשה:

```
Line *lineNewLine(char opcode[], char rd[], char rs[], char rt[], char imm[], int location)
{
Line *new_line = (Line *)malloc(sizeof(Line));
if (new_line != NULL)
{
strcpy(new_line->rt, rt);
strcpy(new_line->rs, rs);
strcpy(new_line->rd, rd);
strcpy(new_line->opcode, opcode);
strcpy(new_line->imm, imm);
new_line->location = location;
new_line->next = NULL;
}
return new_line;
}
```

מציאת השורה מהאינדקס:

```
Line *lineGetByIndex(Line *head, int index)
{
if (index < 0)
{
return NULL;
}
Line *temp = head;
int i;
for (i = 0; i < index; i++)
{
if (temp != NULL)
{
temp = temp->next;
}
else
{
break;
}
}
return temp;
}
```

מציאת שורה ממיקומה:

```
Line *lineGetByLocation(Line *head, int location)
{
Line *temp = head;
int i;
while (temp->location != location)
{
temp = temp->next;
if (temp == NULL)
{
break;
}
}
return temp;
}
```

מציאת השורה האחרונה:

```
Line *lineLast(Line *head)
{
Line *temp = head;
while (temp->next != NULL)
{
temp = temp->next;
}
return temp;
}
```

2.5 Simulator

קובץ זה מכיל את המהלך הכולל של הסימולטור.

פונקציית ה-main של הסימולטור:

```
int main(int argc, char* argv[])
{
//allocating a pointer to each file we would use later
FILE* fp_memin = NULL, * fp_diskin = NULL, * fp_irq2in = NULL, * fp_memout = NULL,
* fp_regout = NULL, * fp_trace = NULL, * fp_hwregtrace = NULL, * fp_cycles = NULL, * fp_leds = NULL, * fp_display7seg = NULL,
* fp_diskout = NULL, * fp_monitor_txt = NULL, * fp_monitor_yuv;
//an array of all file pointers
FILE** file_pointers[] = { NULL, &fp_memin, &fp_diskin, &fp_irq2in, &fp_memout,
&fp_regout, &fp_trace, &fp_hwregtrace, &fp_cycles, &fp_leds, &fp_display7seg, &fp_diskout, &fp_monitor_txt, &fp_monitor_yuv };
//used for saving data for the irq2 handler
int irq2[200];
int disk_cycle = 0;
//used for saving data for the disk handler
int* disk_cycle_ptr = &disk_cycle;
int is_in_task = 0;
const int output_file_index = 4; // All file after this inex are output files
// arrays which would represent the register of the proccessor and the io registers
int regs[NUM_REGS] = { 0 }, ioreg[NUM_IOREGS+5] = { 0 };
char memory[MAX_LINES][LINE_MAX_SIZE];
//setting all lines to 000
reset_memory(memory);
char disk_memory[NUM_SECTORS * NUM_SECTOR_LINES][MAX_DISK_LINE_LEN];
//setting all lines to 000
reset_disk_memory(disk_memory);
int monitor[MONITOR_SIZE * MONITOR_SIZE] = { 0 };
int pc = 0, irq = 0, busy_with_interruption = 0;
if (argc != NUM_COMMANDLINE_PARAMETERS) // check the number command line arguments
{
printf("Error: Incorrect command line arguments number\n");
return 1;
}
int i;
// opening the input files
for (i = 1; i < output_file_index; i++)
{
*file_pointers[i] = fopen(argv[i], "r");
if (*(file_pointers[i]) == NULL)
{
printf("Error: The file %s couldn't open properly", argv[i]);
return -1;
}
}
// opening the output files without yuv file
for (i = i; i < argc - 1; i++)
{
*file_pointers[i] = fopen(argv[i], "w");
if (*(file_pointers[i]) == NULL)
{
printf("Error: The file %s couldn't open properly", argv[i]);
return -1;
}
}
//open yuv in binary
*file_pointers[i] = fopen(argv[i], "wb");
if (*(file_pointers[i]) == NULL)
{
printf("Error: The file %s couldn't open properly", argv[i]);
return -1;
}
//reading the fp_irq2in file to irq2 array
add_irq2(fp_irq2in, irq2);
//reading the memory from the input file into memory
read_memory(fp_memin, memory);
//reading the memory from the input file into disk memory
read_disk_memory(fp_diskin, disk_memory);
//run the instructions
run_instructions(regs, ioreg, file_pointers, memory, &is_in_task, irq2, monitor, disk_memory, disk_cycle_ptr);
//writing to the output files
write_cycles(fp_cycles, ioreg[CLK_REG]);
write_regout(fp_regout, regs);
write_memout(fp_memout, memory);
write_diskout(fp_diskout, disk_memory);
write_monitor_txt(fp_monitor_txt, monitor);
write_monitor_yuv(fp_monitor_yuv, monitor);
//close all opened files
close_pf(file_pointers, NUM_COMMANDLINE_PARAMETERS);
return 0;
}
```

סגירת כל הקבצים:

```
void close_pf(FILE** file_pointers[], int argc)
{
int i;
for (i = 1; i < argc; i++)
{
fclose(*file_pointers[i]);
}
}
```

מעבר למחזור השעון הבא:

```
void next_cycle(int* ioreg, int monitor[], char disk_memory[][MAX_DISK_LINE_LEN],
int* pc_pointer, int* is_in_task, int irq2[], char memory[][LINE_MAX_SIZE], FILE**
file_pointers[], int* disk_cycle_ptr) {
ioreg[CLK_REG] = ioreg[CLK_REG] % 0xffffffff + 1; //update the value of the cycle counter
int led = ioreg[LEDS_REG];
FILE* leds_file = *file_pointers[LEDS];
FILE* display7seg_file = *file_pointers[DISPLAY7SEG];
IO_handler(ioreg, monitor, disk_memory, pc_pointer, is_in_task, irq2, disk_cycle_ptr, mem-
ory, led, leds_file, display7seg_file);
}
```

איפוס הזיכרון (בתחילת הריצה):

```
void reset_memory(char memory[][LINE_MAX_SIZE]) {
int i = 0;
for (i = 0; i < MAX_LINES; i++) {
strcpy(memory[i], "00000");
}
}
```

איפוס זיכרון הדיסק (בתחילת הריצה):

```
void reset_disk_memory(char disk_memory[][MAX_DISK_LINE_LEN]) {
int i = 0;
for (i = 0; i < NUM_SECTORS * NUM_SECTOR_LINES; i++) {
strcpy(disk_memory[i], "00000");
}
}
```

קריאת הזיכרון (בתחילת הריצה):

```
void read_memory(FILE* fp_memin, char memory[][LINE_MAX_SIZE])
{
int pc = 0;
int next_pc = 0;
char curent_inst[LINE_MAX_SIZE];
char imm_line[LINE_MAX_SIZE];
while (fgets(curent_inst, LINE_MAX_SIZE, fp_memin))
{
pc = next_pc;
curent_inst[strcspn(curent_inst, "\r\n")] = '\0'; // remove \n and \r
strcpy(memory[next_pc], curent_inst);
next_pc++;
}
}
```

קריאת הדיסק (בתחילת הריצה):

```
void read_disk_memory(FILE* fp_diskin, char disk_memory[][MAX_DISK_LINE_LEN])
{
int i = 0;
char line[MAX_DISK_LINE_LEN];
while (fgets(line, MAX_DISK_LINE_LEN, fp_diskin))
{
line[strcspn(line, "\r\n")] = '\0'; // remove \n and \r
strcpy(disk_memory[i], line);
i++;
}
}
```

קריאת ההוראה הנוכחית:

```
Instruction* read_instruction(int pc, char memory[][LINE_MAX_SIZE])
{
char* curent_inst = memory[pc];
char* imm_line = memory[pc];
if (instructionTypeFromLine(curent_inst) == I_TYPE)
{
imm_line = memory[pc + 1];
}
return instructionFromLine(curent_inst, imm_line, pc);
}
```

הדפסת מצב הרגיסטרים:

```
void print_reg_state(int pc, int* reg, Instruction* inst)
{
const int reg_num = 16;
char reg_name[][20] = { "zero", "imm", "v0", "a0", "a1", "a2", "a3", "t0", "t1", "t2", "s0",
"s1", "s2", "gp", "sp", "ra" };
printf("PC:%d ", pc);
int i;
for (i = 0; i < reg_num; i++)
{
if (i == inst->rd || i == inst->rs || i == inst->rt) // if the register is used in this
instruction, print it in red
printf("\033[031m");
printf("%s:%d ", reg_name[i], reg[i]);
printf("\033[0m"); //resum non color printing
}
printf("\n");
}
```

כתיבת מחזורי השעון:

```
void write_cycles(FILE* fp_cycles, int* cycles)
{
fprintf(fp_cycles, "%u", cycles);
}
```

כתיבה לקובץ המוצא regout:

```
void write_regout(FILE* fp_regout, int* reg)
{
int R0 = reg[0], R1 = reg[1], R2 = reg[2], R3 = reg[3], R4 = reg[4], R5 = reg[5], R6 =
reg[6], R7 = reg[7], R8 = reg[8], R9 = reg[9], R10 = reg[10], R11 = reg[11], R12 = reg[12],
R13 = reg[13], R14 = reg[14], R15 = reg[15];
fprintf(fp_regout,   "%08X\n%08X\n%08X\n%08X\n%08X\n%08X\n%08X\n%08X\n%08X\n%08X\n%08X\n%08X\n%08X\n",
R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15);
}
```

כתיבת קובץ ה-trace:

```
void write_trace(FILE* fp_trace, int pc,char memory[][LINE_MAX_SIZE], int* regs)
{
int R0 = regs[0], R1 = regs[1], R2 = regs[2], R3 = regs[3], R4 = regs[4], R5 = regs[5], R6
= regs[6], R7 = regs[7], R8 = regs[8], R9 = regs[9], R10 = regs[10], R11 = regs[11], R12 =
regs[12], R13 = regs[13], R14 = regs[14], R15 = regs[15];
fprintf(fp_trace, "%03X %s %08X %08X %08X %08X %08X %08X %08X %08X %08X %08X %08X %08X %08X
%08X %08X %08X\n", pc, memory[pc], R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12,
R13, R14, R15);
}
```

כתיבת קובץ ה-memout:

```
void write_memout(FILE* fp_memout, char memory[][LINE_MAX_SIZE]) {
int i = 0;
char line_str[6];
int line;
for (i = 0; i < MAX_LINES; i++) {
line = strtoul(memory[i], NULL, 16) & 0xFFFFF;
fprintf(fp_memout, "%05X\n", line);
}
}
```

כתיבה למוניטור:

```
void write_monitor_txt(FILE* fp_monitor_txt, int monitor[MONITOR_SIZE * MONITOR_SIZE]) {
int i = 0;
for (i = 0; i < MONITOR_SIZE * MONITOR_SIZE; i++) {
fprintf(fp_monitor_txt, "%02X\n", monitor[i] & 0xFF);
}
}
```

כתיבה לקובץ הבינארי של המוניטור:

```
void write_monitor_yuv(FILE* fp_monitor_yuv, int monitor[MONITOR_SIZE * MONITOR_SIZE]) {
int i = 0;
int chars[2];
for (i = 0; i < MONITOR_SIZE * MONITOR_SIZE; i++) {
fprintf(fp_monitor_yuv, "%c", monitor[i]);
}
}
```

כתיבה לקובץ ה-diskout:

```
void write_diskout(FILE* fp_diskout, char disk_memory[][MAX_DISK_LINE_LEN]) {
int i = 0;
int line;
for (i = 0; i < NUM_SECTORS * NUM_SECTOR_LINES; i++) {
line = strtoul(disk_memory[i], NULL, 16);
fprintf(fp_diskout, "%05X\n", line & 0x000FFFFF);
}
}
```

פענוח ההוראה:

```
void decode_inst(int* regs, int* ioreg, Instruction* inst, char memory[][LINE_MAX_SIZE], int* pc_pointer, int* is_in_task, int irq2[], int monitor[], char
disk_memory[][MAX_DISK_LINE_LEN], FILE** file_pointers[], int* disk_cycle_ptr)
{
int io_target_reg;
switch (inst->opcode)
{
case 0: // add
if (inst->rd <= IMM_REG) // wrting to REG0 or REG IMM
{
break; //dont update target register's value
}
regs[inst->rd] = regs[inst->rs] + regs[inst->rt];
break;
case 1: // sub
if (inst->rd <= IMM_REG) // wrting to REG0 or REG IMM
{
break; //dont update target register's value
}
regs[inst->rd] = regs[inst->rs] - regs[inst->rt];
break;
case 2: // mul
if (inst->rd <= IMM_REG) // wrting to REG0 or REG IMM
{
break; //dont update target register's value
}
regs[inst->rd] = regs[inst->rs] * regs[inst->rt];
break;
case 3: // and
if (inst->rd <= IMM_REG) // wrting to REG0 or REG IMM
{
break; //dont update target register's value
}
regs[inst->rd] = regs[inst->rs] & regs[inst->rt];
break;
case 4: // or
if (inst->rd <= IMM_REG) // wrting to REG0 or REG IMM
{
break; //dont update target register's value
}
regs[inst->rd] = regs[inst->rs] | regs[inst->rt];
break;
case 5: // xor
if (inst->rd <= IMM_REG) // wrting to REG0 or REG IMM
{
break; //dont update target register's value
}
regs[inst->rd] = regs[inst->rs] ^ regs[inst->rt];
break;
case 6: // sll
if (inst->rd <= IMM_REG) // wrting to REG0 or REG IMM
{
break; //dont update target register's value
}
regs[inst->rd] = regs[inst->rs] << regs[inst->rt];
break;
case 7: // sra
if (inst->rd <= IMM_REG) // wrting to REG0 or REG IMM
{
break; //dont update target register's value
}
regs[inst->rd] = regs[inst->rs] >> regs[inst->rt];
break;
case 8: // srl
if (inst->rd <= IMM_REG) // wrting to REG0 or REG IMM
{
break; //dont update target register's value
}
if (regs[inst->rt] < 0) {
regs[inst->rd] = (regs[inst->rs] >> regs[inst->rt]); // if its actualy a left shift
}
else
{
regs[inst->rd] = ((regs[inst->rs] & 0x000FFFFF) >> regs[inst->rt]); // get red of sign extentions
}
break;
case 9: // beq
if (regs[inst->rs] == regs[inst->rt])
*pc_pointer = regs[inst->rd];
break;
case 10: // bne
if (regs[inst->rs] != regs[inst->rt])
*pc_pointer = regs[inst->rd];
else
break;
case 11: // blt
if (regs[inst->rs] < regs[inst->rt])
*pc_pointer = regs[inst->rd];
else
break;
case 12: // bgt
if (regs[inst->rs] > regs[inst->rt])
*pc_pointer = regs[inst->rd];
break;
case 13: // ble
if (regs[inst->rs] <= regs[inst->rt])
*pc_pointer = regs[inst->rd];
break;
case 14: // bge
if (regs[inst->rs] >= regs[inst->rt])
*pc_pointer = regs[inst->rd];
break;
case 15: // jal
regs[inst->rd] = *pc_pointer;
*pc_pointer = regs[inst->rs];
break;
case 16: // lw
regs[inst->rd] = extend_sign(strtoul(memory[regs[inst->rs] + regs[inst->rt]], NULL, 16));
//next_clk;
break;
case 17: // sw
sprintf(memory[regs[inst->rs] + regs[inst->rt]], "%05X", regs[inst->rd]);
//next_clk;
break;
case 18: // reti
*is_in_task = 0;
*pc_pointer = ioreg[IRQRETURN_REG];
break;
case 19: // in
io_target_reg = regs[inst->rs] + regs[inst->rt];
if (inst->rd <= IMM_REG) // wrting to REG0 or REG IMM
{
break; //dont update target register's value
}
regs[inst->rd] = ioreg[io_target_reg];
hwregtrace_write(*file_pointers[HWREGTRACE], ioreg[CLK_REG], inst->opcode == 20, io_target_reg, ioreg[io_target_reg]);
break;
case 20: // out
io_target_reg = regs[inst->rs] + regs[inst->rt];
ioreg[io_target_reg] = regs[inst->rd];
hwregtrace_write(*file_pointers[HWREGTRACE], ioreg[CLK_REG], inst->opcode == 20, io_target_reg, regs[inst->rd]);
break;
case 21: // halt
*pc_pointer = -1;
```

הרצת ההוראה:

```
void run_instructions(int regs[NUM_REGS], int* ioreg, FILE** file_pointers[], char memory[][LINE_MAX_SIZE], int* is_in_task, int irq2[], int monitor[], char disk_memory[][MAX_DISK_LINE_LEN], int* disk_cycle_ptr)
{
FILE* fp_trace = *file_pointers[TRACE];
int pc = 0;
int* pc_pointer = &pc;
int old_pc = pc;
int temp_pc = 0;
Instruction* current_instruction;
while (pc != -1)
{
//fetch instruction from memory
current_instruction = read_instruction(pc, memory);
regs[ZERO_REG] = 0;
regs[IMM_REG] = 0;
if (instructionType(current_instruction) == I_TYPE) regs[IMM_REG] = current_instruction->imm;
//Write Trace
write_trace(*file_pointers[TRACE], *pc_pointer, memory, regs);
pc++;
if (instructionType(current_instruction) == I_TYPE)pc++;
decode_inst(regs, ioreg, current_instruction, memory, pc_pointer, is_in_task, irq2, monitor, disk_memory, file_pointers, disk_cycle_ptr);
next_clk;
if (instructionType(current_instruction) == I_TYPE) next_clk;
if ((current_instruction->opcode == 16) || (current_instruction->opcode == 17)) next_clk;
//if it is sw or lw
}
}
```

3 קבצי האסמבלי

3.1 sort

בקובץ זה מופיע קוד האסמבלי הממיין מערך של 16 מספרים.
השתמשנו באלגוריתם BubbleSort על מנת לבצע מיון, הקובץ מחולק ל-3 פונקציות עיקריות:

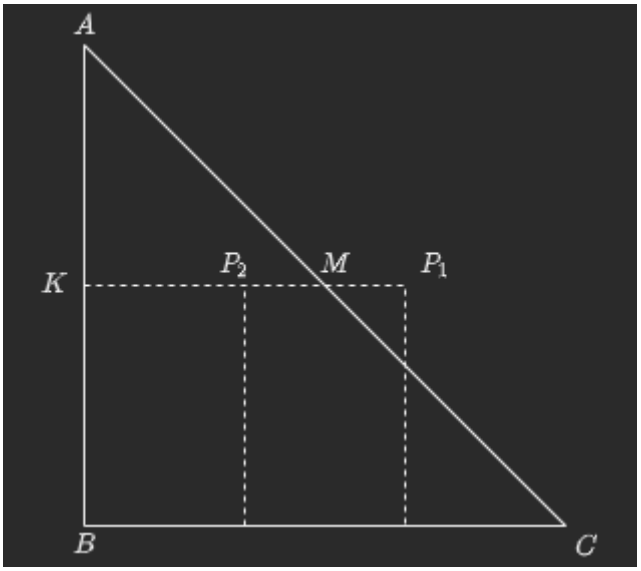
1. פונקציית Swap - מקבלת מהמחסנית את הכתובות של שני הנתונים בזיכרון ומחליפה ביניהם.
2. פונקציית IsSorted - מקבלת מהמחסנית את כתובת ההתחלה של המערך ומחזירה האם הוא ממוין.
3. פונקציית Sort - כל זמן שהמערך לא ממוין בודקת האם $mem[i] > mem[i+1]$, במידה וכן מחליפה ביניהם.

3.2 binom

בקובץ זה מופיע קוד האסמבלי המחשב את תוצאת הנוסחה המתמטית של הבינום של ניוטון באופן רקורסיבי.

3.3 triangle

בקובץ זה מופיע קוד האסמבלי המצייר על המוניטור משולש ישר זווית כאשר נתונים לו שלושת הקודקודים של המשולש.



בהינתן הנקודה P, כאשר ערך ה-y שלה נמצא בין ערך ה-y של A ל-B, ניעזר בנוסחה הבאה:

$$AK \cdot BC \geq KP \cdot AB$$

לקוד האסמבלי יש כמה פונקציות חישוב אורכי הצלעות: AK, BK, BC.
הקוד רץ בין ערכי ה-y של A לערכי ה-y של B (בלולאה החיצונית), בלולאה הפנימית הקוד רץ מערך ה-x של A ומעלה בכל איטרציה את ערך ה-x ב-1. במידה ומתקיים תנאי המשוואה היא מדפיסה את הפיקסל בלבן, אחרת היא קופצת לשורה הבאה.

3.4 disktest

בקובץ זה מופיע קוד האסמבלי הסוכם את 7 הסקטורים הראשונים של הדיסק וכותב את הסכום לסקטור ה-8 של הדיסק.
שלבי הפעולה של סקריפט זה הם:

1. בדיקה האם הדיסק פנוי (אם לא מחכים שיתפנה)
2. בדיקה האם נסכמו 7 סקטורים
3. קריאה מהסקטור
4. בדיקה האם הדיסק פנוי (אם לא מחכים שיתפנה)
5. קריאת התוכן מהזיכרון
6. סכימה עבור כל באפר בזיכרון ושמירה במיקום אחר בזיכרון
7. חזרה להתחלה עד שנסכמו כל הבאפרים בכל הסקטורים
8. כתיבת כל הבאפרים לדיסק
9. בדיקה האם הדיסק פנוי (אם לא מחכים שיתפנה)