Final Project in Computer Structure course 0512.4400 Fall 2022-23

Saleh Yassin 212108187
Ibrahem Howare 322505926
Group num: 59

Hint to eaxaminer:
To run the code extract the folder named: id1_id2, then open the folder fibo, overwrite there the file fib.asm to the SIMP code that you want to check, after that run asm.exe to overwrite memin.txt and then run sim.exe to overwrite memout.txt, regout.txt, trace.txt and cycles.txt.

רקע על הפרויקט:

בפרויקט הזה אנחנו נדרשים לבנות סימוליטור למעבד SIMP שמקבל קוד אסימבלי המכיל שני סוגים של הוראות (I-instruction ו- R-instruction) במבנה הבא:

R format			
19:12	11:8	7:4	3:0
opcode	Rd	rs	rt

I format			
19:12	11:8	7:4	3:0
opcode	rd	rs	rt
imm			

המעבד יכול לבצע הפעולות הבאות:

Opcode Number	Name	Meaning
0	add	R[rd] = R[rs] + R[rt]
1	sub	R[rd] = R[rs] – R[rt]
2	mul	R[rd] = R[rs] * R[rt]
3	and	R[rd] = R[rs] & R[rt]
4	or	R[rd] = R[rs] | R[rt]
5	xor	R[rd] = R[rs] ^ R[rt]
6	sll	R[rd] = R[rs] << R[rt]
7	sra	R[rd] = R[rs] >> R[rt], arithmetic shift with sign extension
8	srl	R[rd] = R[rs] >> R[rt], logical shift
9	beq	if (R[rs] == R[rt]) pc = R[rd]
10	bne	if (R[rs] != R[rt]) pc = R[rd]
11	blt	if (R[rs] < R[rt]) pc = R[rd]
12	bgt	if (R[rs] > R[rt]) pc = R[rd]
13	ble	if (R[rs] <= R[rt]) pc = R[rd]
14	bge	if (R[rs] >= R[rt]) pc = R[rd]
15	jal	R[rd] = next instruction address, pc = R[rs]
16	lw	R[rd] = MEM[R[rs]+R[rt]], with sign extension
17	sw	MEM[R[rs]+R[rt]] = R[rd] (bits 19:0)
18	halt	Halt execution, exit simulator

המעבד משתמש ב-15 רגיסטרים שונים והם:

Register Number	Register Name	Purpose
0	$zero	Constant zero
1	$imm	Sign extended imm
2	$v0	Result value
3	$a0	Argument register
4	$a1	Argument register
5	$a2	Argument register
6	$a3	Argument register
7	$t0	Temporary register
8	$t1	Temporary register
9	$t2	Temporary register
10	$s0	Saved register
11	$s1	Saved register
12	$s2	Saved register
13	$gp	Global pointer (static data)
14	$sp	Stack pointer
15	$ra	Return address

ובנוסף יכול להופיע בקוד שורה מהצורה: **.word address data.**

שמעתיקה ערך "**data**" ישירות ל memory במקום ה- "**address**" .

המעבד מובנה משני חלקים שיש לכל אחד מהם תפקיד אחר שהם ה-"assembler" וה- "simulator":

"assembler":

התפקיד של ה- "assembler" הוא לקרוא את קוד ה SIMP ולתרגם אותו ל- machine code שהוא בעצם מספרים ב- hexadecimal המורכבים מ- 20 ביט (5 באיט), כל הוראה מסוג I תתורגם כשתי שורות (שורה בשביל ה imm) והוראה מסוג R תתורגם כשורה אחת.

ה- "assembler" קולט קובץ fibs.asm (קוד ה- SIMP) ופולט קובץ memin.txt (ה-machine code).

הסבר למה שעשינו בקוד **ה- "assembler"** צעד אחר צעד:

-הגדרנו שלושה סטרקצ'רים אחד מסוג "instruction" המכיל 5 שדות שהם "opcode","rd","rs","rt" ו- "imm" כך ששדה ה-"opcode" מתאים לשני הבאיטים הראשונים ב-"machcode" וכל אחד מהשדות "rd","rs","rt" ו-"imm" מתאים לבאיט השלישי הרביעי והחמישי בהתאם.

סטרקצר מסוג "label" המכיל שתי שדות שהם "name" ו-"address" כך ש "name" מהווה השם של ה-label ו-"address" מהווה המיקום של ה-label ב- memin.

סטרקצר מסוג "word" המכיל שתי שדות שהם "address" ו-"value " כך ש "address" מהווה המיקום ב memin שרוצים להדפיס לו את ה "value".

```
typedef struct { //define instruction as structure
    char opcode[10];
    char rd[5];
    char rs[5];
    char rt[5];
    char imm[25];
}instruction;

typedef struct { //define label as structure
    char name[50];
    int address;

}label;

typedef struct { //define word as structure
    int address;
    int value;

}word;
```

- הגדרנו הפונקציה raedlabel המקבלת את קובץ ה-fipin ומערך labels מסוג label ריק, הפונקציה תעבור על קוד ה- SIMP ותזהה איפה יש label בקוד ותכניס אותו למערך ה-labels עם שדות name ו-address המתאימים.

```
int readlabels(FILE* fipin, label labels[500]) { //function that read the labels and updates the label's
list with labels name and address
    int labelnum = 0;
    int linenumber = 0;
    char line[MAX_LINE];
    int k, j, count = 0;
    while (!feof(fipin)) {
        int flag = 0;
        fgets(line, MAX_LINE, fipin);
        count++;
        if (strcmp(line, "\n") == 0)
            continue;
        if (line[0] == '#') {
            count = count - 1;
            continue;
        }
        if (strstr(line, ".word") != NULL) { //If line is .word, continue
```

```
            count = count - 1;
            continue;
        }
        if (strstr(line, "$imm") != NULL) //If dots are found, this is a label
            count++;
        if (strstr(line, ":") != NULL)
        {
            if (strstr(line, "#") != NULL) // however, ":" can be in a remark. so check for that as well,
if so go to another line
                if ((strstr(line, ":")) > (strstr(line, "#"))) {
                    continue;
                }
            count = count - 1;
            k = 0;
            j = 0;
            do {
                if (line[k] != ':') {
                    if (line[k] != '\t')
                        if (line[k] != ' ')
                        {
                            labels[labelnum].name[j] = line[k];
                            j++;
                        }
                    k++;
                }
            } while (line[k] != ':');
            labels[labelnum].name[j] = '\0';
            labels[labelnum].address = count;
            labelnum++;
        }
    }
    fclose(fipin);
    return 0;
```

}-הגדרנו פונקציה readword המקבלת שורה מקוד ה- SIMP ומזהה אם השורה היא שורת word. ואם כן היא מחזירה סטרקצר מסוג word שמכיל שדות "address" ו-"value" מתאימים.

```
word readword(char line[MAX_LINE]) {
    char* token;
    word wordin;
    char array[3][500];
    printf("%s \n",line);
    token = strtok(line, "\t\n");
    printf("%s \n",token);
    token = strtok(token, " \n");
    printf("%s \n",token);
    for (int i = 0; i < 2; i++) {
        token = strtok(NULL, " \n\t");
        strcpy(array[i], token);
    }
    if (strstr(array[0], "0x")) {
        wordin.address = strtol(array[0], NULL, 16);
    }
    else {
        wordin.address = atoi(array[0]);
    }
    wordin.value = atoi(array[1]);
    return wordin;
}
```

-הגדרנו את הפונקציה indexnumber שמקבלת שם של register או opcode ומחזירה את מספר ה hexadecimal המתאים לו לפי הטבלות שלמעלה.

```
char* indexnumber(char token[MAX_LINE]) {  //function that convert the registers and opcodes to
codes
    if (strstr(token, "$zero") != NULL)
        return "0";
    if (strstr(token, "$imm") != NULL)
        return "1";
```

```
if (strstr(token, "$v0") != NULL)
    return "2";
if (strstr(token, "$a0") != NULL)
    return "3";
if (strstr(token, "$a1") != NULL)
    return "4";
if (strstr(token, "$a2") != NULL)
    return "5";
if (strstr(token, "$a3") != NULL)
    return "6";
if (strstr(token, "$t0") != NULL)
    return "7";
if (strstr(token, "$t1") != NULL)
    return "8";
if (strstr(token, "$t2") != NULL)
    return "9";
if (strstr(token, "$s0") != NULL)
    return "A";
if (strstr(token, "$s1") != NULL)
    return "B";
if (strstr(token, "$s2") != NULL)
    return "C";
if (strstr(token, "$gp") != NULL)
    return "D";
if (strstr(token, "$sp") != NULL)
    return "E";
if (strstr(token, "$ra") != NULL)
    return "F";
if (strstr(token, "add") != NULL)
    return "00";
if (strstr(token, "sub") != NULL)
    return "01";
if (strstr(token, "mul") != NULL)
    return "02";
if (strstr(token, "and") != NULL)
    return "03";
if (strstr(token, "or") != NULL)
    return "04";
if (strstr(token, "xor") != NULL)
    return "05";
if (strstr(token, "sll") != NULL)
    return "06";
if (strstr(token, "sra") != NULL)
    return "07";
if (strstr(token, "srl") != NULL)
    return "08";
if (strstr(token, "beq") != NULL)
    return "09";
if (strstr(token, "bne") != NULL)
    return "0A";
if (strstr(token, "blt") != NULL)
    return "0B";
if (strstr(token, "bgt") != NULL)
    return "0C";
if (strstr(token, "ble") != NULL)
    return "0D";
if (strstr(token, "bge") != NULL)
    return "0E";
if (strstr(token, "jal") != NULL)
    return "0F";
if (strstr(token, "lw") != NULL)
    return "10";
if (strstr(token, "sw") != NULL)
    return "11";
if (strstr(token, "halt") != NULL)
    return "12";
return "";
}
```

-הגדרנו הפונקציה islabel שמזהה אם ה imm הוא label ומחזירה 1 אם כן, אחרת הפונקציה מחזירה 0.

```
Int islabel(char imm[20]) { ///check if $imm is label
    int lab = 0;
    char firstCh[2];
    strncpy(firstCh, &imm[0], 1);
    firstCh[1] = '\0';
    if (firstCh[0] >= 65 && firstCh[0] <= 122)
        lab = 1;
    return lab;
}
```

-הגדרנו שתי פונקציות dectohex ו- dectohexx כך שהפונקציה הראשונה מקבלת מספר decimal בפורם של char ומחזירה אותו ב-hexadecimal, והפונקציה השניה עושה אותו דבר עבור מספרים שלילים.

```
int dectohex(char imm[20], FILE* fp) { //function that convert numbers from char form to
hexadecimal number and print it to memin.txt
    int decimm = atoi(imm);
    char heximm[6];
    sprintf(heximm, "%05X", decimm);;
    fprintf(fp, "%s\n", heximm);
    return 0;
}
int dectohexx(char imm[20], FILE* fp) { //function that convert negative numbers from char to
hexadecimal number and print it to memin.txt
    char lastCh[20];
    strncpy(lastCh, &imm[1], 19);
    int decimm = atoi(lastCh);
    char heximm[9];
    char heximm2[6];
    sprintf(heximm, "%05X", -1 * decimm);;
    strncpy(heximm2, &heximm[3], 5);
    fprintf(fp, "%s\n", &heximm[3]);
    return 0;
}
```

-הגדרנו הפונקציה readinstruction שעושה רוב העבודה באסימפלר, הפונקציה מקבלת קובץ ה-SIMP, מערך ה- labels והקובץ memin, הפונקציה עוברת שורה שורה ומתרגמת הקוד ל machine code ומעתיקה אותו ל- memin.

```
Int readinstructions(FILE* fipin, label labels[500], FILE* fp) { //function that read the
insrucions and print to the memin.txt machcodes and the immediate
    char line[MAX_LINE];
    int wordadd[4096] = { 0 };
    int linenum = 0;
    int finalinst = 0;
    if (fipin == NULL) {
        perror("Error opening file");
        exit(1);
    }
    if (fp == NULL) {
        perror("Error opening file");
        return 1;
    }
    while (!feof(fipin)) {
        int imed = 0;
        char immediate[20];
        // read a command from the assembler file
        fgets(line, MAX_LINE, fipin);
        linenum++;
        if (strcmp(line, "\n") == 0) //If line is blank, continue
```

```
        continue;
    if (line[0] == '#') //If line is Remark, continue
        continue;
    if (strstr(line, ".word") != NULL) { //If line is .word, continue
        char linecpy[MAX_LINE];
        strcpy(linecpy, line);
        word wordin = readword(linecpy);
        if (finalinst <= wordin.address) {
            finalinst = wordin.address;
        }
        linenum = linenum - 1;
        wordadd[wordin.address] = wordin.value;
        continue;
    }
    if (strstr(line, ":") != NULL) {
        linenum = linenum - 1;//If dots are found, this is a label
        continue;
    }
    char* token;
    token = strtok(line, "\t");
    token = strtok(token, ", \n\t");
    char machcode[25] = "";
    for (int I = 0; I < 4; i++) {

        strcat(machcode, indexnumber(token));
        token = strtok(NULL, ", \n\t");
    }
    strcpy(immediate, token);
    if (machcode[2] == '1')
        imed = 1;
    if (machcode[3] == '1')
        imed = 1;
    if (machcode[4] == '1')
        imed = 1;
    if (imed == 1)
    {
        linenum++;
        if (islabel(immediate) == 1) {
            for (int I = 0; I <= 500; i++) {
                //if (strstr(immediate, labels[i].name) != NULL) {
                if (!strcmp(immediate, labels[i].name)) {
                    char value[50];
                    8print(value, "%d", labels[i].address);
                    fprintf(fp, "%s\n", machcode);
                    dectohex(value, fp);
                }
            }
        }

        else if (islabel(immediate) == 0) {
            fprintf(fp, "%s\n", machcode);
            char firstCh[2];
            strncpy(firstCh, &immediate[0], 1);
            if (strstr(firstCh, "-") != NULL) {
                dectohexx(immediate, fp);
            }
            else {
                dectohex(immediate, fp);
            }
        }
    }
    else if (imed == 0)
        fprintf(fp, "%s\n", machcode);
}
for (int I = linenum; I <= finalinst; i++) {
    if (wordadd[i] == 0)
        dectohex("0", fp);
    else {
        char value[50];
```

```
            9print(value, “%d”, wordadd[i]);
            dectohex(value, fp);
        }
    }
    fclose(fipin);
    fclose(fp);
    return 0;

}
```

לבסוף בתוך פונקציית ה- main אתחלנו מערך ה-labels לאפסים, קראנו קובץ הקלט והגדרנו קובץ הפלט ואז קראנו לשתי הפונקציות readlabels ו- readinstructions.

```
void main() {
   label labels[500];
   FILE* fipin = fopen(fibs.txt", "r");
   FILE* fipin1 = fopen("fibs.txt", "r");
   FILE* fp = fopen("memin.txt", "w");
   readlabels(fipin1, labels);
   readinstructions(fipin, labels, fp);
{
```

"simulator":

התפקיד של ה- "simulator" הוא לקבל את ה- machine code שיצא מה-assembler ולעשות לו סימולציה כלומר לבצע הפעולות שנמצאות בו כמו שעושה מעבד ה SIMP.

הסימולטור מקבל קובץ memin.txt (machine code) ומחזיר ארבעה קבצים והם:

cycles.txt: מספר ה-cycles שלקח למעבד כדי לרוץ על הקוד.

regout.txt: ערכי הרגיסטרים אחרי שרצנו על הקוד.

trace.txt: כל שורה מציגה מצב הרגסטרים ב- cycle הנוכחי.

memout.txt: הזיכרון אחרי שסיימנו לרוץ על הקוד.

הסבר למה שעשינו בקוד **ה- "simulator"** צעד אחר צעד:

-הגדרנו סטרקצ'ר אחד מסוג "instruction" המכיל 5 שדות שהם "opcode","rd","rs","rt" ו-"imm" כך ששדה ה-"opcode" מתאים לשני הבאיטים הראשונים ב-"machcode" וכל אחד מהשדות "rd","rs","rt" ו-"imm" מתאים לבאיט השלישי הרביעי והחמישי בהתאם.

```
typedef struct {
    char opcode[10];
    char rd[5];
    char rs[5];
    char rt[5];
}instruction;
```

-הגדרנו הפונקציה strhextointdec שמקבלת מספר ב-hexadecimal בפורמט char והופכת אותו ל int:

```
int strhextointdec(char hex_string[6]) {    //function that turn a hex string to it's decimal value taking
into consideration two's complements
    int hex = strtol(hex_string, NULL, 16); // turn the hex string to dec value
    int dec = 0;                            //initialize the dec value to zero
    if ((hex & 0x80000) > 0) {              // check if the first bit is 1 then the number is negative
        dec = dec - 0x80000;
    }
    dec = dec + (hex & 0x7FFFF);            //get the reset of the values from the first 19 bits
    return dec;
}
```

-הגדרנו הפונקציה loadmem שמקבלת את הקובץ memin ומערך mem של int המאותחל ל-0 וממלא אותו בערכים המתאימים, המערך mem הוא בעצם ה- memory שלנו שבסוף נעביר אותו לקובץ ה- memout:

```
int loadMEM(FILE* memin, int* MEM) { // read a the memin to MEM array
    char line[MAX_LINE];
    int i = 0;
    while (!feof(memin)) {
        fgets(line, MAX_LINE, memin);
        MEM[i] = strhextointdec(line);
        i++;
    }
    return i;
}
```

-הגדרנו הפונקציה moveFP שמחזירה את הפוינטר שקורה הקובץ אל השורה ה-0:

```
void moveFP(FILE* memin, long pc) { //this function move the pointer of the file to the place needed
    char line[MAX_LINE];
    fseek(memin, 0, SEEK_SET); //move the pointer to the needed place
    int pc_help = pc;
    while (pc_help > 0) {
        fgets(line, MAX_LINE, memin);
        pc_help -= 1;
    }
}
```

-הגדרנו הפעולות שהמעבד יכול לבצע, כך שכל פעולה מקבלת ארבעה פוינטרים לרגסטרים pc,rd,rs,rt ובנוסף אם הפעולה היא מסוג branch/jump אזי היא תקבל גם קובץ ה- memin, ואם היא מסוג save/load word אזי היא גם תקבל המערך mem (כי היא צריכה לכתוב אליו או לקרוא ממנו).

```
int add(int rd, int rs, int rt, int* regs, long* pc) {
    regs[rd] = regs[rs] + regs[rt];
    *pc += 1;
    return 0;
}
int sub(int rd, int rs, int rt, int* regs, long* pc) {
    regs[rd] = regs[rs] - regs[rt];
    *pc += 1;
    return 0;
}
int mul(int rd, int rs, int rt, int* regs, long* pc) {
    regs[rd] = regs[rs] * regs[rt];
    *pc += 1;
    return 0;
}
int and(int rd, int rs, int rt, int* regs, long* pc) {
    regs[rd] = regs[rs] & regs[rt];
    *pc += 1;
    return 0;
}
int or(int rd, int rs, int rt, int* regs, long* pc) {
    regs[rd] = regs[rs] | regs[rt];
    *pc += 1;
    return 0;
}
int xor(int rd, int rs, int rt, int* regs, long* pc) {
    regs[rd] = regs[rs] ^ regs[rt];
    *pc += 1;
    return 0;
}
int sll(int rd, int rs, int rt, int* regs, long* pc) {
    regs[rd] = regs[rs] << regs[rt];
    *pc += 1;
    return 0;
}
int sra(int rd, int rs, int rt, int* regs, long* pc) {
    regs[rd] = regs[rs] >> regs[rt];
    *pc += 1;
    return 0;
}
int srl(int rd, int rs, int rt, int* regs, long* pc) {
    int size = sizeof(int);
    if (regs[rs] == 0)
        regs[rd] = regs[rs];
    else
        regs[rd] = (regs[rs] >> regs[rt]) & ~(((regs[rs] >> (size << 3) - 1) << (size << 3) - 1)) >>
(regs[rt] - 1);
    if (rd == 0)
        regs[rd] = 0;
    *pc += 1;
}
int beq(int rd, int rs, int rt, int* regs, long* pc, FILE* file) {
    if (regs[rs] == regs[rt]) {
```

```
        *pc = regs[rd];
        moveFP(file, regs[rd] );
    }
    else {
        *pc += 1;
    }
    return 0;
}
int bne(int rd, int rs, int rt, int* regs, long* pc, FILE* file) {
    if (regs[rs] != regs[rt]) {
        *pc = regs[rd];
        moveFP(file, regs[rd] );
    }
    else {
        *pc += 1;
    }
    return 0;
}
int blt(int rd, int rs, int rt, int* regs, long* pc, FILE* file) {
    if (regs[rs] < regs[rt]) {
        *pc = regs[rd];
        moveFP(file, regs[rd] );
    }
    else {
        *pc += 1;
    }
    return 0;
}
int bgt(int rd, int rs, int rt, int* regs, long* pc, FILE* file) {
    if (regs[rs] > regs[rt]) {
        *pc = regs[rd];
        moveFP(file, regs[rd] );
    }
    else {
        *pc += 1;
    }
    return 0;
}
int ble(int rd, int rs, int rt, int* regs, long* pc, FILE* file) {
    if (regs[rs] <= regs[rt]) {
        *pc = regs[rd];
        moveFP(file, regs[rd] );
    }
    else {
        *pc += 1;
    }
    return 0;
}
int bge(int rd, int rs, int rt, int* regs, long* pc, FILE* file) {
    if (regs[rs] >= regs[rt]) {
        *pc = regs[rd];
        moveFP(file, regs[rd] );
    }
    else {
        *pc += 1;
    }
    return 0;
}
int jal(int rd, int rs, int rt, int* regs, long* pc, FILE* file) {    //jal instruction
    *pc += 1;
    regs[15] = *pc;
    *pc = regs[rs];
    moveFP(file, regs[rs]);
    return 0;
}
int lw(int rd, int rs, int rt, int* regs, long* pc, int* MEM) {
    regs[rd] = MEM[regs[rs] + regs[rt]];
    *pc += 1;
    return 0;
```

```
}
int sw(int rd, int rs, int rt, int* regs, long* pc, int* MEM) {
    MEM[regs[rs] + regs[rt]] = regs[rd];
    *pc += 1;
    return 0;
}
```

-הגדרנו הפונקציה machcodetran שמקבלת מספר hexadecimal בפורמט- char ומחזירה instruction עם שדות מתאימים:

```
instruction machcodetran(char machcode[6]) { //this function split the machine code to opcode ,rd ,rd ,rt
    instruction inst;
    strncpy(inst.opcode, machcode, 2);
    inst.opcode[2] = '\0';
    strncpy(inst.rd, &machcode[2], 1);
    inst.rd[1] = '\0';
    strncpy(inst.rs, &machcode[3], 1);
    inst.rs[1] = '\0';
    strncpy(inst.rt, &machcode[4], 1);
    inst.rt[1] = '\0';
    return inst;
}
```

-הגדרנו הפונקציה iformat שמקבלת ה- machcode ומזהה אם יש בתוכו רגסטר המשתמש ב- imm:

```
int iformat(char machcode[6]) { //the function check if the machine code use an immediate in it
    if (machcode[2] == '1') {
        return 1;
    }
    if (machcode[3] == '1') {
        return 1;
    }
    if (machcode[4] == '1') {
        return 1;
    }
    return 0;
}
```

-הגדרנו הפונקציה readmemin שעושה רוב העבודה בסימולטור, הפונקציה מקבלת פוינטר למערך הרגסטרים (המערך המכיל באינדקס ה- i את הרגסטר ה-i לפי רשימת הרגיסטרים למעלה), ומקבלת את ה- pc (program counter) וחמשת הקבצים (קובץ הקלט memin.txt וארבעת קבצי הפלט (memout.txt, trace,txt, regout.txt, cycles.txt.

הפונקציה קוראת את קובץ הכניסה memin ומחקה למעבד SIMP ומעדכנת קבצי הפלט בהתאם.

```
void meminread(FILE* memin, int* regs, long* pc, int* MEM, long* cycles, FILE* trace, FILE* regout, FILE*
cycles_, FILE* memout) {
    char line[MAX_LINE], linecpy[MAX_LINE];
    char imm[50];
    *pc = 0;
    int i = -1;
    instruction inst;
    if (memin == NULL) {
        perror("Error opening file");
        exit(1);
    }
    while (!feof(memin)) {
        fgets(line, MAX_LINE, memin); //get the machine code from the memory
        char instline[6];
        strncpy(instline, line, 5);
        instline[5] = '\0';
       *cycles += 1;
        if (iformat(line)) { // check if the machine code use an immediate
            *cycles += 1;
            *pc += 1;
            fgets(imm, MAX_LINE, memin); // if the machine code has an immediate read the next line in
memory
            regs[1] = strhextointdec(imm);
```

```
        }
        else {
            regs[1] = 0;
        }
        fprintf(trace, "%03x %s %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x
%08x\n", *pc, instline, regs[0], regs[1], regs[2], regs[3], regs[4], regs[5], regs[6], regs[7], regs[8],
regs[9], regs[10], regs[11], regs[12], regs[13], regs[14], regs[15]);
        inst = machcodetran(line);  // convert the machine code and then check which opcode function does
it have
        if (strhextointdec(inst.opcode) == 0) {
            add(strhextointdec(inst.rd), strhextointdec(inst.rs), strhextointdec(inst.rt), regs, pc);
        }
        if (strhextointdec(inst.opcode) == 1) {
            sub(strhextointdec(inst.rd), strhextointdec(inst.rs), strhextointdec(inst.rt), regs, pc);
        }
        if (strhextointdec(inst.opcode) == 2) {
            mul(strhextointdec(inst.rd), strhextointdec(inst.rs), strhextointdec(inst.rt), regs, pc);
        }
        if (strhextointdec(inst.opcode) == 3) {
            and (strhextointdec(inst.rd), strhextointdec(inst.rs), strhextointdec(inst.rt), regs, pc);
        }
        if (strhextointdec(inst.opcode) == 4) {
            or (strhextointdec(inst.rd), strhextointdec(inst.rs), strhextointdec(inst.rt), regs, pc);
        }
        if (strhextointdec(inst.opcode) == 5) {
            xor (strhextointdec(inst.rd), strhextointdec(inst.rs), strhextointdec(inst.rt), regs, pc);
        }
        if (strhextointdec(inst.opcode) == 6) {
            sll(strhextointdec(inst.rd), strhextointdec(inst.rs), strhextointdec(inst.rt), regs, pc);
        }
        if (strhextointdec(inst.opcode) == 7) {
            sra(strhextointdec(inst.rd), strhextointdec(inst.rs), strhextointdec(inst.rt), regs, pc);
        }
        if (strhextointdec(inst.opcode) == 8) {
            srl(strhextointdec(inst.rd), strhextointdec(inst.rs), strhextointdec(inst.rt), regs, pc);
        }
        if (strhextointdec(inst.opcode) == 9) {
            beq(strhextointdec(inst.rd), strhextointdec(inst.rs), strhextointdec(inst.rt), regs, pc,
memin);
        }
        if (strhextointdec(inst.opcode) == 10) {
            bne(strhextointdec(inst.rd), strhextointdec(inst.rs), strhextointdec(inst.rt), regs, pc,
memin);
        }
        if (strhextointdec(inst.opcode) == 11) {
            blt(strhextointdec(inst.rd), strhextointdec(inst.rs), strhextointdec(inst.rt), regs, pc,
memin);
        }
        if (strhextointdec(inst.opcode) == 12) {
            bgt(strhextointdec(inst.rd), strhextointdec(inst.rs), strhextointdec(inst.rt), regs, pc,
memin);
        }
        if (strhextointdec(inst.opcode) == 13) {
            ble(strhextointdec(inst.rd), strhextointdec(inst.rs), strhextointdec(inst.rt), regs, pc,
memin);
        }
        if (strhextointdec(inst.opcode) == 14) {
            bge(strhextointdec(inst.rd), strhextointdec(inst.rs), strhextointdec(inst.rt), regs, pc,
memin);
        }
        if (strhextointdec(inst.opcode) == 15) {
            jal(strhextointdec(inst.rd), strhextointdec(inst.rs), strhextointdec(inst.rt), regs, pc,
memin);
        }
        if (strhextointdec(inst.opcode) == 16) {
            *cycles += 1;
            lw(strhextointdec(inst.rd), strhextointdec(inst.rs), strhextointdec(inst.rt), regs, pc, MEM);
        }
        if (strhextointdec(inst.opcode) == 17) {
```

```
            *cycles += 1;
            sw(strhextointdec(inst.rd), strhextointdec(inst.rs), strhextointdec(inst.rt), regs, pc, MEM);
        }
        if (strhextointdec(inst.opcode) == 18) {
            break;
        }
    }
    //print the final values in the registers to regout
    fprintf(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%08x", 
regs[2], regs[3], regs[4], regs[5], regs[6], regs[7], regs[8], regs[9], regs[10], regs[11], regs[12], 
regs[13], regs[14], regs[15]);
    //print the number of cycles
    fprintf(cycles_, "%lu", *cycles);
    for (int i = 0; i < 4096; i++) {
        fprintf(memout, "%05X\n", MEM[i] & 0xFFFFF);    // print the memory after runnig all the 
instructions
    }
    fclose(cycles_);  // the next five lines close all the files the code use
    fclose(memin);
    fclose(trace);
    fclose(regout);
    fclose(memout);
}
```

לבסוף בתוך פונקציית ה- main אתחלנו מערך ה- reg_list ו- MEM לאפסים, ואתחלנו את ה- pc ו- cycles, קראנו קובץ הקלט והגדרנו קבצי הפלט ואז קראנו לפונקציה loadMEM שמאתחלת ה MEM ואז קראנו לפונקציה meminread שקוראת קובץ הכניסה memin.txt, מחקה למעבד ה- SIMP, ומעדכנת קבצי היציאה: memout.txt, regout.txt, trace.txt, cycles.txt .

```
void  main() {
    int reg_list[16] = { 0 };
    int MEM[4096] = { 0 };
    long* pc;
    long* cycles = 0;
    FILE* memin = fopen("memin.txt", "r");
    FILE* trace = fopen("trace.txt", "w");
    FILE* regout = fopen("regout.txt", "w");
    FILE* cycles_ = fopen("cycles.txt", "w");
    FILE* memout = fopen("memout.txt", "w");
    int memsize;
    memsize=loadMEM(memin, MEM);
    rewind(memin);
    meminread(memin, reg_list, &pc, MEM, &cycles, trace, regout, cycles_, memout);
{
```