בס"ד

דוקומנטציית פרויקט ISA - מבנה המחשב

נאור שטרית 314619644

שני באטה 207126376

ברק לוי 311431894

אסמבלר

מבני נתונים עיקריים

בחלק זה השתמשנו בעיקר בשני מבני נתונים -

1) Labels - מערך של תוויות (לייבלים) בגודל 4096, כאשר כל תא במערך הוא תווית - struct אשר מכיל את שם התווית ואת הכתובת שמתאימה לתווית זו. בסוף המעבר הראשון על קובץ האסמבלי, הקובץ יכיל את התוויות שנמצאות בקובץ (השם שלהן ואת הכתובת שלהן).
2) Memory - מערך בגודל 4096 של מספרים שלמים, כל תא במערך בעצם מתאים לתא בזיכרון, בסוף ריצת התכנית המערך יכיל את קידוד כל פקודות האסמבלי של הקובץ ואז נעתיק את תוכן המערך לתוך הקובץ memin.txt.

פונקציות עיקריות

read_file – פונקציה זו מקבלת מצביע לקובץ האסמבלי (אשר נפתח בפונקציית ה-main) ואת מספר המעבר שעל הפונקציה לבצע. אם מספר המעבר יהיה 1, הפונקציה תעבור על קובץ האסמבלי ותעדכן את מערך התוויות בהתאם (באמצעות הפונקציה update_labels) , ואם מספר המעבר יהיה 2, אז הפונקציה תעבור על הקובץ ותקודד את הפקודות בקובץ למספרים שלמים ותשמור אותם במערך הזיכרון Memory (באמצעות הפונקציה write_instruction).

get_tokens – מקבלת שורה מהקובץ (לאחר שעבר תיקון ע"י הפונקציה correct_line, אשר מוסיפה רווח אחרי ':' ומורידה רווחים וטאבים לפני ':' אם קיימת תווית בשורה) ומפרקת אותו ל-tokens, כאשר כל token מחזיק חלק מהשורה, לדוגמה: תווית, אופקוד רגיסטרים וקבוע. לאחר שהיא מוצאת כל token היא שומרת אותו במערך של מצביעים שהיא מקבלת.

write_instruction – נקראת עבור כל שורה בקובץ האסמבלי במעבר השני, מקבלת את מערך ה- tokens שמכיל את חלקי השורה, וכותבת למערך Memory את הפקודות המקודדות.

update_labels– נקראת עבור כל שורה בקובץ האסמבלי במעבר הראשון, מקבלת את מערך ה-tokensשמכיל את חלקי השורה, בודקת אם בשורה זו יש תווית אז היא מעדכנת את מערך התוויות עם שם התווית והכתובת שמתאימה לה.

דרך הפיתרון

1. פונקציית ה-main פותחת את קובץ האסמבלי ויוצרת את memin.txt, וקוראת ל-read_file, לשם ביצוע המעבר הראשון.
2. read_file מבצעת את המעבר הראשון על הקובץ, מעדכנת את מערך התוויות וחוזרת לפונקציית ה-main.

3. פונקציית ה-main קוראת ל-read_file , לשם ביצוע המעבר השני.
4. read_file מבצעת את המעבר השני על הקובץ, מעדכנת את מערך הזיכרון וחוזרת לפונקציית ה-main.
5. פונקציית ה-main קוראת לפונקציה write_memory_to_file, אשר מעתיקה את תוכן מערך הזיכרון לתוך הקובץ memin.txt
6. לאחר מכן פונקציית ה-main סוגרת את הקבצים ויוצאת מהתוכנית.

סימולטור

מבני נתונים עיקריים

memory_in –מערך של sring בגודל 4096 אשר מכיל את ערכי הקובץ memin, מתעדכן בהתאם לתוכניות המבוצעות ולאחר מכן מודפס לתוך קובץ הפלט memout .

disk_matrix- מערך של sring בגודל 128*128 אשר מכיל את ערכי הקובץ diskin, כאשר כל סקטור ממוקם אחד אחרי השני לאורך המערך. ערך זה מתעדכן בהתאם לתוכניות המבוצעות ולאחר מכן מודפס לתוך קובץ הפלט diskout .

reg- מערך של int בגודל 16 אשר מכיל את ערכי הרגיסטרים לכל אורך התוכנית ומשתנה בהתאם.

רגיסטרי קלט פלט - רגיסטרים אלו נשמרו בתור משתנים עצמאיים מטעמי נוחות.

פונקציות עיקריות

createTrace- פונקציה זו אחראית על מילוי השורה הבאה בקובץ trace לאחר כל קריאת שורת קוד .

- **decipher_line-** פונקציה זו מפענחת את שורת הקוד ומבצעת אותה. בנוסף, הפונקציה דואגת לקדם את ה-pc בהתאם להוראה שנקראה.

check_disk- פונקציה זו בודקת האם יש שימוש בדיסק ומעדכנת ערכים בהתאם לצורך כמו כן בודקת במידה והדיסק פועל האם הסתיימה פעולת קריאה או כתיבה ומעדכנת בהתאם לכך.

check_timer- פונקציה זו בודקת האם יש שימוש בטיימר ומעדכנת ערכים בהתאם לצורך.

clk_loop-פונקציה זו מעדכנת את השעון בכל מחזור שעון ומאתחלת לאפס במידה והגענו ל 0xffffffff.

createLastFiles-פונקציה זו מדפיסה אל קבצי הפלט את הערכים הרצויים.

דרך הפתרון

- פונקציית הmain פותחת ויוצרת את כל הקבצים הנדרשים.
- קריאת הקובץ memin וכתיבת ההוראות לתוך המערך memory_in.
- קריאת הקובץ diskin וכתיבת ההוראות לתוך המערך dusk_matrix.
- שולפת מקובץ irq2in את זמן הפסיקה הראשון.

ריצה על קוד תוכנית האסמבלר:

- בדיקת הפסיקות- האם יש צורך לקפוץ אליה וביצוע הקפיצה בהתאם.
- קריאת שורה מערך memory_in .
- פענוח והשמת ערך הקבוע imm וקריאה לפונקציה createTrace.
- קריאה לשאר הפונקציות: decipher_line, clk_loop, check_timer,check_disk .
- לאחר שיוצאים מה-while קוראים לפונקציה createLastFiles.
- וסוגרים את כל הקבצים הפתוחים.

תוכניות קוד האסמבלי

1. $summat.asm$:

תוכנית הsummat מורכבת מ3 פונקציות :

1. פונקצית main שבה הכנסנו את ערכי המספרים במטריצות B,A לכתובות המתאימות, הגדלנו את המחסנית (כלפי מטה) ושמרנו בה את הרגיסטרים השמורים.
2. פונקצית הfor שבה ביצענו את הדרוש – עברנו בעזרת לולאת for איבר איבר ב2 המטריצות במקביל, חיברנו את ערכי המספרים ושמרנו בכתובת המתאימה למטריצה השלישית C - מטריצת הסכום.
3. פונקצית END בה שיחררנו את המקום שהקצנו בתחילת התוכנית במחסנית וסיימנו את הריצה בעזרת halt.

2. $bubble.asm$:

בתוכנית זו מימשנו מיון בועה קלאסי בקוד האסמבלי. תחילה נדגיש שמבחינת אופטימיזציה סיימנו את ריצת הפונקציה כאשר במעבר על שלל האיברים (בלולאה הפנימית) במערך לא התבצע ולו $swap$ בודד. זהו תנאי עצירה שיכול לחסוך איטרציות מיותרות.

המיון, כאמור התבצע בעזרת שתי לולאות מקוננות. בתחילת הקוד שמרנו כארגומנט את כתובת האיבר הראשון במערך ומספר האיברים בו. נוסיף כי טענו לזיכרון ערכים רנדומליים ע"י פקודת $word$. בכתובות $1024 - 1039$ וככלל 16 איברים.

פעולתה העיקרית של התוכנית הינה בלייבלים :

$\boldsymbol{OuterLoop}$: הלולאה החיצונית במיון.

$\boldsymbol{InnerLoop}$: הלולאה הפנימית שמחפשת מתי לבצע $swap$

$\boldsymbol{swap}$ מבצע שיחלוף בין שני איברים במערך (על פי גודלם).

$\boldsymbol{Condition}$ למעשה תנאי "אם" שבודק האם עלינו לצאת ללולאה החיצונית או שמא נוכל לסיים את המיון

בסיום התוכנית כמובן ששחררנו זיכרון במחסנית (שנשמרה לפי הקונבנציה כלפי מטה).

3. $qsort.asm$:

תוכנית הqsort מורכבת מ7 פונקציות :

1. פונקצית main שבה הכנסנו את ערכי המספרים של המערך לכתובות המתאימות, הגדלנו את המחסנית (כלפי מטה), שמרנו בה את הרגיסטרים השמורים וביצענו קריאה לפונקצית quicksort. כאשר מסיימים וחוזרים לפונקציה, פונקצית המaim מסיימת את הריצה בעזרת halt.
2. פונקצית הquicksort שבה איחסנו את הרגיסטרים הדרושים ועברנו לפונקצית partition במידה ותנאי הif שבאלגוריתם מתקיים. אחרת, סיימנו.
3. פונקצית הpartition מוצאת את ערך הpivot הדרוש באלגוריתם והיא עושה זאת בעזרת פונקצית הpivot.
4. פונקצית הpivot שחוזרת על עצמה כלולאה עד שמגיעה לערך הדרוש, בהתאם לאלגוריתם אותו אנו מיישמים, וכאשר מגיע לערך זה היא עוברת לפונקצית loop.
5. פונקצית הloop מבצעת את עיקר אלגוריתם הquicksort בעזרת קירות רקורסיביות לquicksort וקריאות לפונקצית הswap.

6. פונקצית swap מבצעת את החילוף בין 2 ערכים שאותם צריך להחליף כדי להתקדם במיון המערך וחוזרת לאחר מכן לפונקצית swap.
7. פונקצית END בה שיחררנו את המקום שהקצנו בתחילת התוכנית במחסנית.

4. *leds.asm*:

תוכנית הleds מורכבת מ4 פונקציות:

פונקצית main שבה ביצענו קריאה לפונקצית leds. כאשר מסיימים וחוזרים לפונקציה, פונקצית המaim מסיימת את הריצה בעזרת halt.

פונקצית הleds שמגדירה את הtimermax , מדליקה את הנורה הראשונה (נורת הLSB) ואת timerenable.

פונקצית הonesec שדואגת לכך שבין נורה לנורה תעבור דקה שלמה.

פונקצית הloop שאליה מגיעים אחרי שעוברת הדקה שספרנו בonesec והיא מכבה את הנורה הנוכחית, מקדמת אותנו לנורה הבאה ומדליקה אותה. במידה והנורה הנוכחית היא האחרונה, הפונקציה מחזירה את הערך לפונקציה הראשית ומגיעה לhalt שבmain.

5. *clock.asm*:

```
#####################################
# $to store the current time        #
# $t1 arithmetic usage              #
# $t2 indicating on 9 - corner case #
#####################################
```

על מנת להבין טוב יותר את תוכנית הclock הוספנו את הגדרת המשתנים הללו.
שמרנו בזכרון שני נתונים מיוחדים שעזרו לנו לבצע את פונקציה זו. האחד: שעת התחלת השעון כפי שנתבקש. השני, מקרה הקצה של מעבר מהשנייה האחרונה של השעה 19 אל השנייה הראשונה לשעה 20. זאת משום שהפונקציה מממשת אינקרמנטציה באחד לשניות ולכן נוצר מקרה קצה עבור שנייה זו. עתה ניתן להבין בצורה ברורה יותר מדוע בחרנו להשתמש ב $t2$ כמשתנה שמייצג את הספרה 9, זוהי למעשה הפעם היחידה בתוכנית שספרה זו תיהיה ה LSB בהצגת השעון. נדע לעבור לשעה 20 ששמורה במקום 1025 בזכרון ע"י כך שנדע מתי אנחנו בשעה 19:59:59. הלייבילים העקרים במימוש התוכנית, לראייה:

- **Main**: בשורות אלה למעשה בצענו את השמירה של הערכים המיוחדים שמרנו לתוך $timermax$ את הערך 254 וזאת משום שאנחנו רוצים שיחלפו בדיוק 256 מחזורי שעון (בדיוק שנייה לפי המעבד הנ"ל) בין הצגה להצגה של השעון.
המימוש השתמש בלופ של 254 שורות – סרק שכופות קידום ועוד 2 שורות אפשור הצג על מנת להשלים בידיוק 256 מחזורי שעון – הלוא שנייה והצגה של השעה הבאה.
- **clockRun**: מאפשרים את המסך להתחיל בשעה ההתחלתית דואגים לעלות את $timerenable$.
- **CoolDown**: כפיית קידום של מחזורי שעון ובדיקה האם הגענו ל$timermax$ כלומר עברנו 254 מחזורי שעון מאז שהצגנו את השעה האחרונה בשעון.
- **Display**: מתקדמים 2 מחזורי שעון אחרי הcooldown ומאפשרים את השעה הרצויה בצג. סה"כ 256 מחזורי שעון. עורכים בדיקה כמובן למקרי הקצה שלנו, האם הLSB הינו הספרה 9. את הבדיקה הזו אנחנו עושים עם מסכה של 15 ופעולת$bitwise\ and$. כלומר ארבעה ביטים

ימניים דלוקים . וזאת משום שהספרה 9 בייצוג האקססה שלה הוא ארבעה ביטים גם כן. במידה וכך הדבר נקפוץ לליבל הבא :

- $\boldsymbol{CornerCase}$: בלייבל זה נטען את השעה 20 כך שנוכל להציג אותה בפעם הבאה.

6. $disktest.asm$:

```
##########################################
# $t0 = sector number                    #
# $t1 = read/write mode to diskcmd       #
# $t2 == 1  - indicating mode if(t1 != t2)  #
# $t3 = diskstatus                       #
##########################################
```

את רעיון תוכנית הדיסק אפשר להבין בצורה ברורה יותר ע"י תיאור ארבעת הרגיסטרים הללו.

ממעוף הציפור, פעולת התוכנית מתחילה מקריאה של סקטור מספר 0, ולאחר 1024 מחזורי שעון כותבת אותו למקום $\$s0 + 4$ וזאת משום שסקטור 0 מועתק לסקטור 3, סקטור 1 מועתק לסקטור 4 וכן הלאה. לכן אחרי כל פעולת קריאה נקדם את מספר הסקטור בארבע ואחרי פעולת כתיבה נוריד ממנו 3. $\$t1$ הוא משתנה אינדיקטור שנודע להבחין בפעולה האחרונה שנעשית – קריאה או כתיבה ולפיה נדע שהפעולה הבאה היא הפעולה ההפוך מהאחרונה שהתבצעה.

נסביר זאת באמצעות הלייבלים הראשיים בפונקציה :

- **DiskReadWrite** בלייבל זה נעדכן את ה$disksector$ הרלוונטי ונקבע את הפעולה עבורו – קריאה\ כתיבה ע"י עדכון מתאים של $diskcmd$.
- **Cool – down** בלייבל זה נחכה לאות שהסימולטור פולט $diskstatus$ (לאחר 1024 מחזורי שעון) ע"י קפיצה חוזרת ונשיניית לליבל זה (כופים למעשה קידום של מחזורי שעון).
- **Reading** בתחילה נבדוק את הפעולה הקודמת שהייתה על הדיסק, במידה והיא הייתה כתיבה, נמשיך בלייבל זה ונקרא מהמקום הרצוי בזכרון כאשר נקדם את ה-$sector - number$ בקבוע של 4. ונחזור לליבל של $DiskReadWrite$אחרת, נקפוץ לליבל שמבצע את פעולת הכתיבה.
- $\boldsymbol{Writing}$ נבצע הכנה לפעולת הכתיבה ועדכון משתנים רלונטים. בין היתר נחסר בקבוע 3 את מספר הסקטור שבפעולה הבאה נקרא ממנו.

קוד תוכנית האסמבלר :

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <stdlib.h>

#include <string.h>
#include <ctype.h>


#define TRUE 1
#define FALSE 0

#define MAX_LINES 4096
#define MAX_LINE_LEN 501 // +1 for '\0'
#define MAX_LABEL_LEN 51 // +1 for '\0'
#define MAX_TOKENS_IN_LINE 6

#define OPCODES { "add", "sub", "and" ,"or", "sll" , "sra" , "srl", "beq" ,
"bne" , "blt" , "bgt" , "ble" , "bge", "jal", "lw", "sw", "reti" , "in", "out"
,"halt" , ".word" }
// all values' indices match their opcode number!

#define OPCODES_LEN 21 // the length of the array above

#define REGS { "$zero", "$imm", "$v0", "$a0" ,"$a1", "$t0", "$t1", "$t2",
"$t3", "$s0", "$s1", "$s2", "$gp", "$sp", "$fp", "$ra"}
// all registers' indices match their number!
#define REGS_LEN 16 // the length of the array above


typedef       struct label {

       char name[MAX_LABEL_LEN];
       int address;

} Label;


void read_file(FILE* asm_file, int pass_num);
void write_memory_to_file(FILE* out_file);
void write_instruction(char* tokens[], int PC);

int check_value(char* line_start);
int get_reg(char* reg);

void correct_line(char* line, char*corrected_line);
void get_tokens(char* line, char* tokens[]);

int get_imm(char* str);

int update_PC(char* tokens[]);
void update_labels(char* tokens[], int PC);

int is_equal_str(char* str1, char* str2);
int str2int(char* str);


Label Labels[MAX_LINES]; // stores labels name and addresses
int label_index = 0; //first empty label index in labels array
```

```c
int Memory[MAX_LINES] = { 0 }; // represents the memory, in the end will be
written to memin.txt
int mem_index = 0; // current empty slot in memory array
int mem_end = 0; // store index of the last non-zero slot


int main(int argc, char** argv)
{

	FILE * ASM_file = NULL;
	FILE * MEMIN = NULL;

	if (argc < 2)
	{
		printf("Arg Amount Error\n");
		return 1;
	}

	ASM_file = fopen(argv[1], "r");
	MEMIN = fopen("memin.txt", "w");

	if (ASM_file == NULL || MEMIN == NULL)
	{
		printf("Couldn't open file, terminating process\n");
		return 1;
	}

	// read file, perform first pass
	read_file(ASM_file, 1);

	// read file, perform second pass
	read_file(ASM_file, 2);

	// write from memory array to memin.txt
	write_memory_to_file(MEMIN);

	fclose(ASM_file);
	fclose(MEMIN);

	return 0;
}


// performs first pass and second pass over the code
// pass_num = 1 -> first pass  -> get labels      (get the labels and their
addresses)
// pass_num = 2 -> second pass -> write to memory (write instructions)
void read_file(FILE* asm_file, int pass_num)
{
	char line[MAX_LINE_LEN]; // holds line from file
	char corrected_line[MAX_LINE_LEN + 1]; // holds line after correction,
+1 for the additional whitespace

	char* tokens[MAX_TOKENS_IN_LINE]; // tokens, will hold parts of the line

	int PC = 0;                           // current PC

	while (fgets(line, MAX_LINE_LEN, asm_file) != NULL) // read file line by
line
	{
		correct_line(line, corrected_line);
```

```
		get_tokens(corrected_line, tokens);

		// first pass - get labels
		if (pass_num == 1)
			update_labels(tokens, PC);

		// second pass - get instructions in hex and write in file
		else
			write_instruction(tokens, PC);

		// update Program Counter
		PC += update_PC(tokens);

	}
	rewind(asm_file); // Returns the pointer to the beginning of the file
}


// writes to out_file the contents of the memory array
void write_memory_to_file(FILE* out_file)
{
	for (int i = 0; i < mem_end; i++) {

		fprintf(out_file, "%08X\n", Memory[i] & 0xffffffff); // "&
0xffffffff " is for dealing with negative numbers
	}

}

// corrects line from file if there is no ' ' after ':', so we won't read the
label and opcode as one token
// and removes whitespaces between label name and ':'
void correct_line(char* line, char*corrected_line)
{
	strcpy(corrected_line, line);

	char* colon_index1 = strchr(corrected_line, ':'); // check if there is
':' in line
	if (colon_index1 != NULL) // if there is -> make sure there are no
whitespaces before it and at least one whitespace after it
	{
		while (*(colon_index1 - 1) == ' ' || *(colon_index1 - 1) == '\t')
// make sure there are no whitespaces before ':'
		{
			*(colon_index1 - 1) = ':';
			*(colon_index1) = ' ';
			colon_index1--;
		}

		if (*(colon_index1 + 1) != ' ' || *(colon_index1 + 1) != '\t') //
make sure there is at least one whitespaces after ':'
		{
			*(colon_index1 + 1) = ' ';
			char* colon_index2 = strchr(line, ':');

			strcpy(colon_index1 + 2, colon_index2 + 1);
		}
	}
}


// recieves tokens of the line and returns how much we should add to the PC
```

```
int update_PC(char* tokens[])
{
	int first_token = check_value(tokens[0]);
	int opcode;
	int rd_index = 2; // token index of the first register in the
instruction

	if (first_token == -2) return 0; // check if empty line

	if (first_token == -1)  // check if first token is label
	{
		opcode = check_value(tokens[1]); // first token is label ->
second one is opcode if exists

		if (opcode == -2)  // check if second token is NULL
			return 0;
	}
	else  // first token is opcode
	{
		opcode = first_token;
		rd_index = 1;
	}
	if (opcode == 20) return 0; // if opcode is ".word"

	return 1;

}


// splits line into tokens
void get_tokens(char* line, char* tokens[])
{
	char* delimiters = " \n\t,";    // line delimiters


	char* number_sign_index = NULL;

	int is_comment_start = FALSE;

	int tokens_index = 0;

	char* token = strtok(line, delimiters);

	while (token != NULL && *token != '#') // check if token is not NULL and
doesnt begin with '#'
	{
		number_sign_index = strchr(token, '#'); // number_sign - '#',
holds address of '#' in token if exists

		if (number_sign_index != NULL)  // check if token has '#'
		{
			*number_sign_index = '\0';  // truncate token
			is_comment_start = TRUE;    // indicates that after
current token a comment has started
		}

		tokens[tokens_index] = token;
		tokens_index++;

		if (is_comment_start) // if a comment has started -> no more
tokens
			break;
```

```
		token = strtok(NULL, delimiters); // read next token
	}

	for (int i = tokens_index; i < MAX_TOKENS_IN_LINE; i++) // set rest
tokens to NULL
		tokens[i] = NULL;
}


// updates the labels array
void update_labels(char* tokens[], int PC)
{
	if (check_value(tokens[0]) != -1) // check if label is present in the
line
		return;

	strcpy(Labels[label_index].name, tokens[0]); //add label to the array
	Labels[label_index].address = PC;

	int i = 0;
	while (Labels[label_index].name[i] != ':')
		i++;

	Labels[label_index].name[i] = '\0';

	label_index++;
}


// converts instruction to hex and writes to Memory array
void write_instruction(char* tokens[], int PC)
{
	int opcode, rd, rs, rt, imm;
	int first_token, index_offset = 0;
	int use_imm = FALSE;

	first_token = check_value(tokens[0]);

	if (first_token == -2) return; // check if empty line

	if (first_token == -1)  // check if first token is label
	{
		index_offset = 1; // instruction starts after first token

		int second_token = check_value(tokens[1]);

		if (second_token == -2)  // check if second token is NULL
			return;
	}

	opcode = check_value(tokens[0 + index_offset]);

	if (opcode == 20) // if opcode is ".word"
	{
		int address = str2int(tokens[1 + index_offset]); // convert
address to int
		int data = str2int(tokens[2 + index_offset]);    // convert data
to int

		Memory[address] = data;
```

```
		if (address + 1 > mem_end)
			mem_end = address + 1;
		return;
	}

	rd = get_reg(tokens[1 + index_offset]);
	rs = get_reg(tokens[2 + index_offset]);
	rt = get_reg(tokens[3 + index_offset]);

	imm = get_imm(tokens[4 + index_offset]);
	if (imm < 0)rt++;

	Memory[mem_index] = opcode * 16 * 16 * 16 * 16 * 16 * 16 + rd * 16 * 16
* 16 * 16 * 16 + rs * 16 * 16 * 16 * 16 + rt * 16 * 16 * 16 + imm;
	mem_index++;

	if (mem_index > mem_end)
		mem_end = mem_index;
}

// converts imm field in instruction to the appropriate value
int get_imm(char* str)
{
	// check if imm is label
	for (int i = 0; i < label_index; i++) // check if imm is label, if it is
return address
		if (strcmp(str, Labels[i].name) == 0)
			return Labels[i].address;

	return str2int(str);
}


// a line starts with label, an opcode or ".word"
// the function recieves the first word in the line and returns:
// its opcode number if its an opcode,
// 20 if it's ".word",
// -1 if it's a label and -2 if it's NULL
int check_value(char* line_start)
{
	if (line_start == NULL) return -2;

	char* has_colon = strchr(line_start, ':'); // checks if token has the
':' at the end of it

	if (has_colon != NULL) return -1;

	char* opcodes[] = OPCODES;

	for (int i = 0; i < OPCODES_LEN; i++)
		if (is_equal_str(line_start, opcodes[i]))
			return i;

	return -3; // shouldn't get here
}


// recieves register str, returns register number
int get_reg(char* reg)
{
	char* regs[] = REGS;
```

```
	// check which register reg is
	if (is_equal_str(reg, "$0")) return  0;
	for (int i = 0; i < REGS_LEN; i++)
		if (is_equal_str(reg, regs[i]))
			return i;

	return -1; // shouldn't get here
}


// compares lower case version of two strings,
// if equal returns TRUE, else FALSE
int is_equal_str(char* str1, char* str2)
{
	int i = 0;
	while (str1[i] != '\0' || str2[i] != '\0')
	{
		if (tolower(str1[i]) != tolower(str2[i])) // convert to lower
case and compare
			return FALSE;
		i++;
	}
	if (str1[i] != '\0' || str2[i] != '\0') // if strings are of diffrent
length they are not equal
		return FALSE;

	return TRUE;
}


// gets str of a number returns its integer value
int str2int(char* str)
{
	if (str[0] == '0' && (str[1] == 'x' || (str[1] == 'X'))) // check if str
is hex
		return strtol(str, NULL, 0);
	else // str is decimal
		return strtol(str, NULL, 10);
}
```

קוד תוכנית הסימולטור:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define TRUE 1
#define FALSE 0


#define MAX_LINES 4096
#define SIZE 9
#define REG_NUM 16
#define SECTORS_NUM 128
#define SECTOR_SIZE 128


int irq2_cycle = -1;
char disk_matrix[SECTORS_NUM*SECTOR_SIZE][SIZE];
char memory_in[MAX_LINES][SIZE];
int max_line_disk = 0;
int irq = 0;
int irq_flg = FALSE;
int disk_counter_cycles = 0;
int total_cycles = 0;
int pc = 0;
int max_line_mem_counter = 0;
int reg[REG_NUM] = { 0 };
char line[SIZE];
char name_ioreg[15];

//IO Registers
unsigned int irq0enable = 0;
unsigned int irq1enable = 0;
unsigned int irq2enable = 0;
unsigned int irq0status = 0;
unsigned int irq1status = 0;
unsigned int irq2status = 0;
unsigned int irqhandler = 0;
unsigned int irqreturn = 0;
unsigned long clks = 0;
unsigned int leds = 0;
unsigned int display = 0;
unsigned int timerenable = 0;
unsigned long timercurrent = 0;
unsigned int timermax = 0;
unsigned int diskcmd = 0;
unsigned int disksector = 0;
unsigned int diskbuffer = 0;
unsigned int diskstatus = 0;


int opcode;
int rd;
int rs;
int rt;
int imm;

FILE* memin, *diskin, *irq2in, *memout, *regout, *trace, *hwregtrace, *cycles,
*ledsF, *displayF, *diskout;
```

```c
/*Utility func*/
int getHex3(char* source);
int getHex8(char* source);
int hex2int(char ch);
int getAddress(int address);

/*Create func*/
void createTrace();
void createLastFiles();


/*Line Func*/
void add();
void sub();
void andf();
void orf();
void sll();
void sra();
void srl();
void beq();
void bne();
void blt();
void bgt();
void ble();
void bge();
void jal();
void lw();
void sw();
void reti();
void in();
void out();
void halt();

void clk_loop();
void check_disk();
void check_timer();
void decipher_line();


int main(int argc, const char* argv[])
{
	if (argc < 4)//open and check files
	{
		printf("Arg Amount Error");
		return 0;
	}

	memin = fopen(argv[1], "r");
	diskin = fopen(argv[2], "r");
	irq2in = fopen(argv[3], "r");
	memout = fopen("memout.txt", "w");
	regout = fopen("regout.txt", "w");
	trace = fopen("trace.txt", "w");
	hwregtrace = fopen("hwregtrace.txt", "w");
	cycles = fopen("cycles.txt", "w");
	ledsF = fopen("leds.txt", "w");
	displayF = fopen("display.txt", "w");
	diskout = fopen("diskout.txt", "w");
	if (memin == NULL || diskin == NULL || irq2in == NULL || memout == NULL
		|| regout == NULL || trace == NULL || hwregtrace == NULL ||
cycles == NULL
```

```
		|| ledsF == NULL || displayF == NULL || diskout == NULL)
	{
		printf("FILE Error");
		return 0;
	}

	while (!feof(memin))//memory_in  is filled
	{
		int check;
		if (check = fscanf(memin, "%s", memory_in[max_line_mem_counter])
!= 0)
		{
			max_line_mem_counter++;
		}
	}

	fclose(memin);

	while (!feof(diskin))//disk is filled
	{
		int check;
		if (check = fscanf(diskin, "%s", disk_matrix[max_line_disk]) !=
0)
			max_line_disk++;
	}
	fclose(diskin);

	if (!feof(irq2in))//update the first irq2
		fscanf(irq2in, "%d", &irq2_cycle);



	while (pc > -1)//Starting the program
	{
		irq = ((irq0enable & irq0status) | (irq1enable & irq1status) |
(irq2enable & irq2status));
		if (irq == 1 && irq_flg == FALSE)// check irq
		{
			irq_flg = TRUE;
			irqreturn = pc;
			pc = irqhandler;
		}

		if (irq2_cycle == total_cycles) //check if we need to change
irq2status
		{
			irq2status = 1;
			if (!feof(irq2in))//update the next irq2
				fscanf(irq2in, "%d", &irq2_cycle);
		}


		strcpy(line, memory_in[pc]);

		char str[4] = { line[5], line[6], line[7],'\0' };
		imm = getHex3(str);
		reg[1] = imm;

		createTrace();
		decipher_line();
		check_disk();
		check_timer();
```

```
		clk_loop();
		total_cycles++;


	}

	createLastFiles();

	fclose(memout);
	fclose(regout);
	fclose(trace);
	fclose(hwregtrace);
	fclose(cycles);
	fclose(ledsF);
	fclose(displayF);
	fclose(diskout);

	fclose(irq2in);
}

void createLastFiles()//Creates the output files
{
	for (int i = 2; i < REG_NUM; i++)//print regout file
	{
		fprintf(regout, "%08X\n", reg[i]);
	}
	for (int i = 0; i < max_line_mem_counter; i++)//print memout
	{
		int instruction = getHex8(memory_in[i]);
		if (instruction < 0)
			instruction -= 0xFFFFFFFF00000000;
		fprintf(memout, "%08X\n", instruction);
	}
	fprintf(cycles, "%ld\n", total_cycles);


	// updates into diskout file the matrix 'disk-out-matrix'
	for (int i = 0; i < SECTORS_NUM * SECTOR_SIZE; i++)
	{
		fprintf(diskout, "%s\n", disk_matrix[i]);
	}

}

void clk_loop() //Updates the clock as needed
{
	if (clks == 0xFFFFFFFF)
		clks = 0;
	else clks++;
}

void createTrace()//Creates the file trace
{
	fprintf(trace, "%08X %s ", pc, line);
	for (int i = 0; i < REG_NUM - 1; i++)
		fprintf(trace, "%08x ", reg[i]);

	fprintf(trace, "%08x\n", reg[REG_NUM - 1]);///to print the last one
without tab
}
```

```
void decipher_line() //decipher the line
{

	opcode = hex2int(line[0]) * 16 + hex2int(line[1]);
	rd = hex2int(line[2]);
	rs = hex2int(line[3]);
	rt = hex2int(line[4]);



switch (opcode)
	{
	case 0:		add(); break; //The opcode is add
	case 1: sub(); break;	//The opcode is sub
	case 2: andf(); break;	 //The opcode is and
	case 3: orf(); break;	//The opcode is or
	case 4: sll(); break;	//The opcode is sll
	case 5: sra(); break;	//The opcode is sra
	case 6: srl(); break;	//The opcode is srl
	case 7: beq(); break;	//The opcode is beq
	case 8: bne(); break;	//The opcode is ben
	case 9: blt(); break;	//The opcode is blt
	case 10: bgt(); break;	 //The opcode is bgt
	case 11: ble(); break;	 //The opcode is ble
	case 12: bge(); break;	 //The opcode is bge
	case 13: jal(); break;	 //The opcode is jal
	case 14: lw(); break;	//The opcode is lw
	case 15: sw(); break;	//The opcode is sw
	case 16: reti(); break;	//The opcode is reti
	case 17: in(); break;	//The opcode is in
	case 18: out(); break;	//The opcode is out
	case 19: halt(); break;	//The opcode is halt

	default:;

	}
	pc += 1;
	reg[0] = 0;	//Ensures that Register 0 has not changed
}
//Functions of operations
void add()
{
	reg[rd] = reg[rs] + reg[rt];
}
void sub()
{
	reg[rd] = reg[rs] - reg[rt];
}
void andf()
{
	reg[rd] = reg[rs] & reg[rt];
}
void orf()
{
	reg[rd] = reg[rs] | reg[rt];
}
void sll()
{
	reg[rd] = reg[rs] << reg[rt];
}
void sra()
{
```

```
        reg[rd] = reg[rs] >> reg[rt];

}

void srl() {
        reg[rd] = (int)((unsigned)reg[rs] >> reg[rt]);

}
void beq() {
        if (reg[rs] == reg[rt])
                pc = (reg[rd] & 0x00000FFF) - 1;
}
void bne() {
        if (reg[rs] != reg[rt])
                pc = (reg[rd] & 0x00000FFF) - 1;

}
void blt() {
        if (reg[rs] < reg[rt])
                pc = (reg[rd] & 0x00000FFF) - 1;

}
void bgt() {
        if (reg[rs] > reg[rt])
                pc = (reg[rd] & 0x00000FFF) - 1;

}
void ble() {
        if (reg[rs] <= reg[rt])
                pc = (reg[rd] & 0x00000FFF) - 1;

}
void bge() {
        if (reg[rs] >= reg[rt])
                pc = (reg[rd] & 0x00000FFF) - 1;

}

void jal() {
        reg[15] = pc + 1;
        pc = (reg[rd] & 0x00000FFF) - 1;
}

void lw()
{
        char  ln1[SIZE];
        int address = getAddress(reg[rs] + reg[rt]);
        strcpy(ln1, memory_in[address]);
        reg[rd] = getHex8(ln1);
}

void sw()
{
        int address = getAddress(reg[rs] + reg[rt]);
        char ch[SIZE];
        sprintf(ch, "%08x", reg[rd]);
        strcpy(memory_in[address], ch);
        if (max_line_mem_counter < address)
                max_line_mem_counter = address;

}
```

```
void reti() {
	pc = irqreturn - 1;
	irq_flg = FALSE;
}

void in() {
	int rst = reg[rs] + reg[rt];
	switch (rst)
	{
	case 0: reg[rd] = irq0enable;	strcpy(name_ioreg, "irq0enable");
break;
	case 1: reg[rd] = irq1enable;	strcpy(name_ioreg, "irq1enable");
break;
	case 2: reg[rd] = irq2enable;	strcpy(name_ioreg, "irq2enable");
break;
	case 3: reg[rd] = irq0status;	strcpy(name_ioreg, "irq0status");
break;
	case 4: reg[rd] = irq1status;	strcpy(name_ioreg, "irq1status");
break;
	case 5: reg[rd] = irq2status;	strcpy(name_ioreg, "irq2status");
break;
	case 6: reg[rd] = irqhandler;	strcpy(name_ioreg, "irqhandler");
break;
	case 7: reg[rd] = irqreturn;	strcpy(name_ioreg, "irqreturn");
break;
	case 8: reg[rd] = clks;		strcpy(name_ioreg, "clks");
break;
	case 9: reg[rd] = leds;		strcpy(name_ioreg, "leds");
break;
	case 10: reg[rd] = display;	strcpy(name_ioreg, "display");
break;
	case 11: reg[rd] = timerenable;	strcpy(name_ioreg, "timerenable");
break;
	case 12: reg[rd] = timercurrent; strcpy(name_ioreg, "timercurrent");
break;
	case 13: reg[rd] = timermax;	strcpy(name_ioreg, "timermax");
break;
	case 14:reg[rd] = diskcmd;	strcpy(name_ioreg, "diskcmd");
break;
	case 15:reg[rd] = disksector;	strcpy(name_ioreg, "disksector");
break;
	case 16:reg[rd] = diskbuffer;	strcpy(name_ioreg, "diskbuffer");
break;
	case 17:reg[rd] = diskstatus;	strcpy(name_ioreg, "diskstatus");
break;
	default: strcpy(name_ioreg, "Error");

	}
	fprintf(hwregtrace, "%ld READ %s %08x\n", total_cycles, name_ioreg,
reg[rd]);
	reg[0] = 0;	//Ensures that Register 0 has not changed
}

void out() {
	int rst = reg[rs] + reg[rt];
	switch (rst)
	{
	case 0:  irq0enable = (unsigned)reg[rd] & 0x00000001;
strcpy(name_ioreg, "irq0enable");	break;
	case 1:  irq1enable = (unsigned)reg[rd] & 0x00000001;
strcpy(name_ioreg, "irq1enable");	break;
```

```
        case 2:  irq2enable = (unsigned)reg[rd] & 0x00000001;
strcpy(name_ioreg, "irq2enable");   break;
        case 3:  irq0status = (unsigned)reg[rd] & 0x00000001;
strcpy(name_ioreg, "irq0status");   break;
        case 4:  irq1status = (unsigned)reg[rd] & 0x00000001;
strcpy(name_ioreg, "irq1status");   break;
        case 5:  irq2status = (unsigned)reg[rd] & 0x00000001;
strcpy(name_ioreg, "irq2status");   break;
        case 6:  irqhandler = (unsigned)reg[rd] & 0x00000fff;
strcpy(name_ioreg, "irqhandler");   break;
        case 7:  irqreturn = (unsigned)reg[rd] & 0x00000fff;
strcpy(name_ioreg, "irqreturn");    break;
        case 8:  printf("we cant change it");
break;//we cant change it
        case 9:  leds = (unsigned)reg[rd];            strcpy(name_ioreg,
"leds");
                fprintf(ledsF, "%ld %08x\n", total_cycles, reg[rd]);
                break;//update leds file
        case 10: display = (unsigned)reg[rd];     strcpy(name_ioreg, "display");
                fprintf(displayF, "%ld %08x\n", total_cycles, reg[rd]);
                break;//update display file
        case 11: timerenable = (unsigned)reg[rd] & 0x00000001;
strcpy(name_ioreg, "timerenable");  break;
        case 12: timercurrent = (unsigned)reg[rd]; strcpy(name_ioreg,
"timercurrent"); break;
        case 13: timermax = (unsigned)reg[rd];    strcpy(name_ioreg,
"timermax");     break;
        case 14: if (diskstatus == 0)
        {
                diskcmd = (unsigned)reg[rd] & 0x00000003;
                diskstatus = 1;
                if (diskcmd == 3) diskcmd = 0;

                if (diskcmd == 2)//write
                        for (int i = 0; i < SECTOR_SIZE; i++)
                                strcpy(disk_matrix[disksector*SECTOR_SIZE + i],
memory_in[diskbuffer + i]);

                if (diskcmd == 1)//read
                        for (int i = 0; i < SECTOR_SIZE; i++)
                                strcpy(memory_in[diskbuffer + i],
disk_matrix[disksector*SECTOR_SIZE + i]);
        }
                         else return;//if diskstatus==1 -dont do anything and dont
print in hwregtrace
                strcpy(name_ioreg, "diskcmd");      break;

        case 15: if (diskstatus == 0)
                disksector = (unsigned)reg[rd] & 0x0000007f;
                         else return;//if diskstatus==1 -dont do anything and dont
print in hwregtrace
                strcpy(name_ioreg, "disksector");   break;

        case 16: if (diskstatus == 0)
                diskbuffer = (unsigned)reg[rd] & 0x00000fff;
                         else   return;//if diskstatus==1 -dont do anything and
dont print in hwregtrace
                strcpy(name_ioreg, "diskbuffer");   break;

        case 17: diskstatus = (unsigned)reg[rd] & 0x00000001;
                strcpy(name_ioreg, "diskstatus");  break;
```

```
        default: strcpy(name_ioreg, "Error");
        }
        fprintf(hwregtrace, "%ld WRITE %s %08x\n", total_cycles, name_ioreg,
reg[rd]);
}

void halt()
{
        pc = -10;
}

int getHex3(char* source) //Converts three hexadecimal characters to a number
{
        int n = (int)strtol(source, NULL, 16);
        if (n > 0x7ff)
                n -= 0x1000;
        return n;
}
int getHex8(char* source)//Converts eight hexadecimal characters to a number
{
        int n = (int)strtoul(source, NULL, 16);
        if (n > 0x7fffffff)
                n -= 0x100000000;
        return n;
}
int hex2int(char ch)//Hexadecimal character converter to number
{
        if (ch >= '0' && ch <= '9')
                return ch - '0';
        if (ch >= 'A' && ch <= 'F')
                return ch - 'A' + 10;
        if (ch >= 'a' && ch <= 'f')
                return ch - 'a' + 10;
        printf("Char given, %c, is invalid - non hex character.\n", ch);
        return -1;
}

int getAddress(int address)//Checks if the address is valid
{
        if (address < 0)
        {
                printf("Address given, %X, is invalid - negative.\n", address);
                return -10;
        }

        if (address >= 4096)
        {
                printf("Address given, %X, is invalid - exceeds limited space in
memory.\n", address);
                address = address & 0x0FFF; //if given address is too high, take
only 12 LSBs.
                printf("Simulator refers only to 12 LSBs, %X in this case. \n",
address);
        }
        return address;
}

void check_disk()//Checks and performs disk operations
{

        if (disk_counter_cycles == 1024)//finish disk read/write
        {
```

```
                irq1status = 1;
                disk_counter_cycles = 0;
                diskstatus = 0;
                diskcmd = 0;
        }

        if (diskstatus == 1)// disk_counter_cycles++ if need
                disk_counter_cycles++;
}
void check_timer()//Checks and performs timer operations
{
        if (timerenable == 1)
        {
                if (timermax == timercurrent)
                {
                        irq0status = 1;
                        timercurrent = 0;
                }
                else timercurrent++;
        }

}
```

קוד תוכניות האסמבלי:

1. *summat.asm* :

```
main:
        .word 0x100 2                   #values of matrice 1= A
        .word 0x101 4
        .word 0x102 6
        .word 0x103 8
        .word 0x104 10
        .word 0x105 12
        .word 0x106 14
        .word 0x107 16
        .word 0x108 18
        .word 0x109 20
        .word 0x10A 22
        .word 0x10B 24
        .word 0x10C 26
        .word 0x10D 28
        .word 0x10E 30
        .word 0x10F 32
        .word 0x110 1                   #values of matrice 2 = B
        .word 0x111 3
        .word 0x112 5
        .word 0x113 7
        .word 0x114 9
        .word 0x115 11
        .word 0x116 13
        .word 0x117 15
        .word 0x118 17
        .word 0x119 19
        .word 0x11A 21
        .word 0x11B 23
        .word 0x11C 25
        .word 0x11D 27
        .word 0x11E 29
        .word 0x11F 31


        add $sp , $zero , $imm, 0x130
        add $t0 , $zero , $imm, -3      # t0 = -3
        add $sp , $sp , $t0, 0          # sp = sp - 3 (define space in stack)
        sw $s0 , $sp , $imm, 0          # store s0 in stack
        sw $s1 , $sp , $imm, 1          # store s1 in stack
        sw $s2 , $sp , $imm, 2          # store s2 in stack

        add $s0 , $zero , $imm , 0x100  # set $s0 to the address of the first argument of matrix A.
        add $s1 , $zero , $imm , 0x110  # set $s1 to the address of the first argument of matrix B.
        add $s2 , $zero , $imm , 0x120  # set $s2 to the address of the first argument of matrix C - the sum matrix.
```

```
for:
        lw $t0 , $s0 , $zero, $zero         # t0 = A[s0]
        lw $t1 , $s1 , $zero     , $zero    # t1 = B[s1]
        add $t0 , $t0 , $t1 , $zero         # t0 = t0+t1 = A[s0] + B[s1]
        sw $t0 , $s2 , $zero , $zero        # save result to the matrix.

        add $s2 , $s2 , $imm , 1            # s2 is now pointing to the next argument of matrix C.
        add $s1 , $s1 , $imm , 1            # s1 is now pointing to the next argument of matrix B.
        add $s0 , $s0 , $imm , 1            # s0 is now pointing to the next argument of matrix A.

        add $t0 , $s0, $imm , 0             # $t0 = $s0
        add $t1 , $zero , $imm , 0x110      #$t1=0x110
        ble $imm, $t0, $t1, for             #if we didnt done: go again to for with the next arguments
        bgt $imm, $t0, $t1, END             #else - go to END

END:
        lw $s0 , $sp , $imm, 0              # restore values from stack
        lw $s1 , $sp , $imm, 1              # restore values from stack
        lw $s2 , $sp , $imm, 2              # restore values from stack
        add $sp , $sp , $imm , 3            # sp = sp + 3 (release space in stack).
        halt  $zero, $zero , $zero ,0       #halt
```

:bubble.asm .2

```
            add $sp , $zero , $imm , 500
            add $a0 , $zero , $imm ,1024            # loading the address of the first element of the array
            add $a1 , $zero , $imm , 16             # loading the number of elements in the array
            jal  $imm , $zero , $zero , BubbleSort  # unconditional jump to BubbleSort
            halt $zero, $zero, $zero, 0
BubbleSort:
            add  $sp, $sp, $imm, -6                 # space for 6 slots
            sw   $s0 , $sp , $zero, 0
            sw   $s1 , $sp , $imm , 1
            sw   $s2 , $sp , $imm , 2
            sw   $a0 , $sp , $imm , 3
            sw   $a1 , $sp , $imm , 4
            sw   $v0 , $sp , $imm , 5
            sub  $a1 , $a1 , $imm , 1               # $a1 = 15
OuterLoop:                                          # if zero swapps in inner loop - break and return
            add $v0 , $zero , $imm , 0              # $v0 stores: 0 if no swaps 1 otherwise
            add $s2 , $zero , $imm, -1              # set j = -1
InnerLoop:
            add $s2 , $s2 , $imm , 1                # j++
            bge $imm , $s2 , $a1 , Condition        # checking if j>=15
            add $t0 , $a0 , $s2, 0                  # array + j
            lw  $s0 , $t0 , $imm , 0                # $s0 = array[j]
            lw  $s1 , $t0 , $imm , 1                # $s1 = array[j+1]
            bgt $imm, $s0 , $s1 , Swap              # if(array[j] > array[j+1])
            beq $imm , $zero , $zero , InnerLoop    # unconditional jump
Swap:
            sw  $s1 , $t0 , $imm , 0                 #set array[j+1] = array[j]
            sw  $s0 , $t0 , $imm , 1                 #set A[j] = A[j+1]
            add $v0 , $zero , $imm, 1                #set v0 = 1 (true) in other words, swap occured
            beq $imm , $zero, $zero, InnerLoop       #unconditional jump
Condition:
            bne $imm , $v0 , $zero , OuterLoop       #if swaps occurred back to outerLoop
            beq $imm , $zero , $zero , Finish        #if no swaps - function execution can stop ---> break
Finish:
            lw   $s0 , $sp , $zero , 0               # restore $s0 from stack
            lw   $s1 , $sp , $zero , 1               # restore $s1 from stack
            lw   $s2 , $sp , $zero , 2               # restore $s2 from stack
            lw   $a0 , $sp , $zero , 3               # restore array address
            lw   $a1 , $sp , $zero , 4               # restore argument - number of elements
            lw   $v0 , $sp , $zero , 5               # restore return address
            add  $sp , $sp , $imm, 6                 # release allocated space
            beq  $ra, $zero, $zero, 0                # return fron function
           .word 1024 2
           .word 1025 -56
           .word 1026 1
           .word 1027 10
           .word 1028 -6
           .word 1029 4
           .word 1030 94
           .word 1031 12
           .word 1032 -1
           .word 1033 0
           .word 1034 712
           .word 1035 222
           .word 1036 3334
           .word 1037 -333
           .word 1038 543
           .word 1039 225
```

```
main:
        .word 1024 12                       # set A[0] = 12
        .word 1025 6                        # set A[1] = 6
        .word 1026 5                        # set A[2] = 5
        .word 1027 11                       # set A[3] = 11
        .word 1028 20                       # set A[4] = 20
        .word 1029 23                       # set A[5] = 23
        .word 1030 34                       # set A[6] = 34
        .word 1031 34                       # set A[7] = 34
        .word 1032 -3                       # set A[8] = -3
        .word 1033 22                       # set A[9] = 22
        .word 1034 0                        # set A[10] = 0
        .word 1035 8                        # set A[11] = 8
        .word 1036 -1                       # set A[12] =-1
        .word 1037 2                        # set A[13] = 2
        .word 1038 40                       # set A[14] = 40
        .word 1039 1                        # set A[15] = 1
        add $sp , $zero , $imm, 1022
        add $s0 , $zero , $imm, 1024        # the address of the start of the array - A into $s0.
        add $a0 , $zero , $imm, 0           # low = 0
        add $a1 , $zero , $imm, 15          # high = 15
        jal $imm, $zero , $zero , quicksort # go to quicksort
        halt $zero, $zero, $zero, 0         # halt execution
```

```
quicksort:
        add $sp, $sp, $imm, -3              # space in stack
        sw $a0, $sp, $imm, 0                # store a0
        sw $a1, $sp, $imm, 1                # store a1
        sw $ra, $sp, $imm, 2                # return address

        blt $imm, $a0, $a1, partition       # if low < high : go to partition
        beq $imm, $zero, $zero, END         #  else : go to END

partition:
        add $t1, $a0, $imm, -1              # t1 = i
        add $t2, $a1, $imm, 1               # t2 = j

pivot:
        lw $t0, $s0, $a0, 0                 # $t0=pivot
        add $t2, $t2, $imm, -1              # $t2 = $t2 -1
        lw $t3, $t2, $s0, 0                 #  $t3=A[j]
        bgt $imm, $t3, $t0, pivot           # if A[j]>pivot : go again to "pivot"

loop:
        add $t1, $t1, $imm, 1               # $t1 = $t1 +1
        lw $t3, $s0, $t1, 0                 # $t0=A[i]
        blt $imm, $t3, $t0, loop            # if A[j]>=pivot : go agin to loop
        blt $imm, $t1, $t2, swap            # else: if i<j : go to swap with A[j],A[I]
        add $a1, $t2, $imm, 0               # $a1= $t2= j
        jal $imm, $zero, $zero, quicksort   # go to quicksort with A, p, j
        lw $a1, $sp, $imm, 1                # restore $a1 = r
        add $a0, $t2, $imm, 1               # $a0 = j+1
        jal $imm, $zero, $zero, quicksort   # go to quicksort with A, j+1, r

END:
        lw $a0, $sp, $imm, 0                # release s0
        lw $a1, $sp, $imm, 1                # release a0
        lw $ra, $sp, $imm, 2                # release a1
        add $sp, $sp, $imm, 3               # restore the stack
        beq $ra, $zero, $zero, 0            # return $ra

swap:
        lw $t0, $t2, $s0, 0                 # t0 = A[j]
        sw $t3, $t2, $s0, 0                 # A[j] = A[i]
        sw $t0, $t1, $s0, 0                 # A[i] = A[j]
        beq  $imm, $zero, $zero, pivot      # go back to "pivot"
```

:leds.asm .4

```
main:
        jal $imm, $zero, $zero, leds          # start the func leds
        halt $zero, $zero, $zero, 0           #halt execution
leds:
        add $t1, $zero, $imm, 255             # $t1 = 255
        out $t1, $zero, $imm, 13              # timermax = 252

        add $t2, $zero, $imm, 1               # $t2 = 1
        out $t2, $zero, $imm, 11              # turn on the timerenable
        out $t2, $zero, $imm, 9               # turn on the first led

onesec:
        in $t0, $zero, $imm, 3                # turn on irq0status
        beq  $imm, $t0, $zero, onesec         # if $t0 = irq0status  = 0 : go again to "onesec",
                                                                  else : continue to loop

loop:
        out $zero, $zero,  $imm, 3            # turn off irq0status
        add $t2, $t2, $t2, $zero              # $t1 = 2 * $t1
        out $t2, $zero, $imm, 9               # turn on the next led
        beq $ra, $zero, $t2, 0                # if $t2 = 0 : finish
        add $zero, $zero, $zero, 0            # we need 256 cycles between 2 leds
        beq  $imm, $zero, $zero, onesec       # go back to "onesec"  : for the 1 second we need
```

:clock.asm .5

```
"""""""""""""""""""""""""""""""""""""
Main:           add $t2 , $zero , $imm , 9           # $s2 = 9
                .word 1024 0x195955                  # starting time
                .word 1025 0x1fffff                  # one second before the switch of 200000
                add $t0 , $zero , $imm , 254         # $t0 =  254
                out $t0 , $zero , $imm , 13          # timermax = 254 clock cycles
                jal $imm , $zero , $zero , clockRun  # (unconditional jump) starting the function of clock counting from 19:59:00 to 20:00:05
                halt $zero , $zero , $zero , 0       # finishing clock-test
clockRun:
                lw  $t0 , $zero , $imm , 1024        # storing the starting time (19:59:55)to $t0
                out $t0 , $zero , $imm , 10          # enabling the display
                add $t1 , $zero , $imm , 1           # $t1 = 1
                out $t1 , $zero , $imm , 11          # timerenable = 1 (turning in timer-enable)
CoolDown:
                in  $t1  , $zero  , $imm , 3         # $t1 = irq0status = ---> 0: when timercurrent = timermax, timercurrent : otherwise
                beq $imm, $t1, $zero, CoolDown       # staying in this loop until timermax reached.
Display:
                out $zero, $zero, $imm, 3            # irq0status = 0
                add $t0 , $t0 , $imm , 1             # next second increment
                out $t0 , $zero , $imm , 10          # display the next second. here we finish 256 clock cycles which equally to 1 sec
                and $t1 , $t0 , $imm , 15            # masking the 4 LSB bits of the time with 15 mask (bin(15)=000..->..1111)
                beq $imm , $t1 , $t2 , CornerCase    # fixing corner case of 19:59:59 - the only time last digit is in clock should be 9
                and $t1 , $t0 , $imm , 15            # masking the 4 LSB bits with 1's mask
                beq $ra , $t1 , $imm 5               # cheacking if the last digit(4LSB's) is 5 if yes break.
                beq $imm , $zero , $zero , CoolDown  # back to count cpu cycles in order to accomplish 1 second.
CornerCase:
                lw  $t0 , $zero , $imm , 1025        # loading the time second before 200000
                beq $imm , $zero , $zero , CoolDown  # return from Interrunpt
```

.6 ***: disktest.asm***

```
Main:
                add $a0 , $zero , $imm , 1024                 # $a0 = 1024
                out $a0 , $zero , $imm , 16                       #setting diskbuffer to 1024
                add $t2 , $zero , $imm , 1                    # $t2 = 1
                jal $imm , $zero , $zero , DiskInitialized    #jump to diskfunction
                halt $zero , $zero , $zero , 0                #finish program
DiskInitialized:
                add $t0 , $zero , $zero , 0                       # $t0=0 - sector number initialized to 0
                add $t1 , $zero , $imm ,1                     # $t1=1 - initialized for reading mode
DiskReadWrite:
                out $t0 , $zero , $imm , 15                   #setting disksector
                out $t1 , $zero , $imm , 14                   #setting diskcmd as $t1
Cool-down:
                in  $t3  , $zero , $imm , 17                  #set $t3 = diskstatus
                bne $imm , $zero , $t3 , Cool-down            #continue looping until diskstatus = 0
                out $zero, $imm, $zero, 1                     # turn off irq1status
Reading:
                add $t2 , $zero , $imm , 1                    # $t2 = 1
                bne $imm , $t1 , $t2 , Writing                # jump to the Reading-switch epoch becasue the last diskcmd
                                                              # was writing -> bench if the last diskcmd was writing
                add $t1 , $zero , $imm , 2                    # $t1 = 2 next time writing mode
                add $t0 , $t0 , $imm , 4                      # $t0+=4
                beq $imm , $zero , $zero , DiskReadWrite      # return to DiskReadWrite for next operation
Writing:
                add $t1 , $zero , $imm , 1                    # $t1 = 1 indicating on reading mode
                sub $t0 , $t0 , $imm , 3                      # $t0-=3
                beq $ra , $t0 , $imm , 4                      # if we reached to read from sector number 4 - break!
                   beq $imm , $zero , $zero , DiskReadWrite       # return to DiskReadWrite for next operation
```