מבנה המחשב - דוקומנטציה לפרויקט ISA

רון טרבלוס 308536341

יותם כרמי 305194193

מתן אחיאל 205642119

בפרויקט זה מימשנו פונקציונליות של $Assembler$ ו-$Simulator$ עבור מעבד $RISC$ בשם $SIMP$ אשר מסוגל לטפל בשגרת פסיקות. בנוסף, כתבנו תוכניות בשפת $Assembly$לבדיקת המעבד.

אסמבלר:

1. ראשית, הגדרנו גדלים קבועים לפי הדרישות הנתונות בפרויקט (גודל הזיכרון הראשי, אורך שורה ואורך לייבל מקסימליים):

```
//Constants Definitions
#define MEMORY_SIZE 4096
#define MAX_LINE_LENGTH 500
#define MAX_LABEL_LENGTH 50
```

2. הגדרנו מבנה נתונים בשם label שתפקידו לשמור את השם והמיקום של כל הלייבלים אשר נמצאים בקובץ הבדיקה שמקבל האסמבלר:

```
//Data structure of the labels.

typedef struct {
        char label_name[MAX_LABEL_LENGTH + 1];
        int location;
        } label
```

3. בנינו מבנה נתונים בשם instruction_by_fields שמטרתו לשמור את החלקים השונים אשר יש בכל שורת אסמבלי.

```
//Data structure of instruction, seperate by fields.
typedef struct {
        char *label;
        char *opcode;
        char *rd;
        char *rs;
        char *rt;
        char *imm;
        } instruction_by_fields;
```

4. הגדרנו מבנה נתונים בשם line_type שמטרתו לסווג את סוג השורה שהתקבלה על מנת שנדע כיצד לטפל בה בהמשך:

```
//Define line type.
typedef enum {
	EMPTY,
	LABEL,
	LABEL_REGULAR, //label and regular instruction in the same one line
	WORD,
	REGULAR, //opcode, rd, rs, rt, imm
	} line_type;
```

5. כמו כן בתחילת הקוד ביצענו Declaration לכל הפונקציות הקיימות באסמבלר לפני ה-main:

```
//Decleration of all functions in the program.
instruction_by_fields parse_line(char *line);
line_type determine_line_type(instruction_by_fields ins);
void execute(FILE *input_file, FILE *output_file);
void first_pass(FILE *input_file);
int second_pass(FILE *input_file);
int opcode_to_number(char *opcode);
int register_to_number(char *reg);
int immediate_to_number(char *imm);
int encode_instruction(int opcode, int rd, int rs, int rt, int imm);
int HexToInt2sComp(char * h);
int HexCharToInt(char h);
```

6. הגדרנו מספר משתני עזר שתפקידם מצוין בהערה (או נובע ישירות מהשם):

```
label labels[MEMORY_SIZE]; //global array to store the labels and their location.
int labels_amount = 0;
int memory[MEMORY_SIZE]; //global array to store the memory
int label_regular_counter = 0;
```

<u>הסבר על הפונקציות:</u>

```
instruction_by_fields parse_line(char *line);
```

פונקציה זו מקבלת שורה מקובץ ה-asm ומפצלת אותה לרכיביה השונים:
label,opcode,rd,rs,rt,imm. הפונקציה מבצעת השמה למשתנה מסוג instruction_by_fields ומחזירה אותו.

```
line_type determine_line_type(instruction_by_fields ins);
```

פונקציה זו מקבלת משתנה מסוג instruction_by_fields ובודקת את סוג השורה, כאשר האפשרויות הקיימות הן : שורה רגילה, שורה ריקה, לייבל, word, לייבל והוראה יחד באותה השורה.

```
void first_pass(FILE *input_file);
```
פונקציה זו מקבלת מצביע לקובץ אסמבלי ומעדכנת במערך lables את כל הלייבלים אשר קיימים בקוד.

```
int second_pass(FILE *input_file);
```
פונקציה זו עוברת על קוד האסמבלי בפעם השנייה ומבצעת תרגום של הקוד לשפת מכונה.

```
int opcode_to_number(char *opcode);
int register_to_number(char *reg);
int immediate_to_number(char *imm);
```
הפונקציות מקבלות מצביע כפי שהשם מתאר ומחזירות ערך מספרי לפי המפורט בפרויקט.

```
int encode_instruction(int opcode, int rd, int rs, int rt, int imm);
```
פונקציה זו מקבלת את החלקים השונים של ההוראה ומרכיבה אותם לכדי הוראה שלמה בגודל 32 ביטים.

```
int HexToInt2sComp(char * h);
```
פונקציה זו מבצעת הפיכה של מספר המיוצג בהקסדצימלי למספר בייצוג המשלים ל-2.

```
int HexCharToInt(char h);
```
פונקציה זו מקבלת אות (חוקית) של ייצוג הקסדצימלית או מספר (כ-char) ומחזירה את הערך המספרי כ-int.

```
void execute(FILE *input_file, FILE *output_file);
```
זוהי הפונקציה אשר מוציאה לפועל את כל הפעולות שהאסמבלר מבצע. היא מאגדת בתוכה את כל הפונקציות האחרות ונקראת ישירות מה-main. היא מקבלת כקלט את קובץ תוכנית האסמבלי ומוציאה כפלט את תמונת הזיכרון הראשי memin.

<u>דרך הפעולה של האסמבלר:</u>

בשלב הראשון האסמבלר פותח שני קבצים:

א. קובץ קלט של תוכנית הבדיקה (program.asm)- מכיל קוד אסמבלי.

ב. קובץ הפלט של תמונת הזיכרון (memin.txt)- בסוף פעולת האסמבלר זוהי תמונת הזיכרון הראשי לפני ביצוע הפקודות. קובץ זה הינו אחד מקבצי הקלט של הסימולטור.

לאחר פתיחת הקבצים יש בדיקה שלא התרחשה שגיאה בזמן פתיחת הקבצים, ובמידה והכל תקין ממשיכים לפונקציית execute אשר עוברת על קוד האסמבלי בשתי ריצות:

ריצה ראשונה באמצעות הפונקציה first_pass אשר שומרת את השמות והכתובות של הלייבלים לתוך מערך. הריצה השנייה מתבצעת באמצעות הפונקציה second_pass אשר בה יתבצע התרגום של שפת האסמבלי לשפת המכונה.

על מנת להפריד כל שורה לרכיבים השונים משתמשים בפונקציה parse_line אשר ממירה אותה למשתנה ins שהוא טיפוס מסוג instruction_by_fields.

בשלב הבא מתבצעת בדיקה באמצעות הפונקציה line_type שמטרתה לסווג את סוג השורה. כך, כאשר מדובר בשורה מסוג word נבצע את ההוראה ונכניס את הערך לתוך הזיכרון במקום המתאים באופן ישיר. במידה והשורה הינה מסוג LABEL_REGULAR או REGULAR, ממירים את כל הרכיבים של ההוראה למספרים לפי הטבלאות אשר מפורטות בתרגיל.

כל המרכיבים של ההוראה מתחברים לכדי הוראה שלמה על ידי הפונקציה encode_instruction ובסיומה הפקודה תמוקם במקום הייעודי בזיכרון הראשי.

סימולטור:

תפקיד הסימולטור הוא לסמלץ את לולאת ה fetch-decode-execute של המעבד. הסימולטור מקבל את הקובץ memin.txt שנוצר על ידי האסמבלר ומכיל את הוראות האסמבלר בשפת מכונה, ותפקידו לבצע את הפקודות ולשנות את הזיכרון בהתאם.

הסימולטור מקבל את קבצי הקלט הבאים באמצעות שורת הפקודה:

Memin.txt , diskin.txt, irq2in.txt

ופולט בסוף התוכנית את קבצי הפלט הבאים:

Memout.txt ,regout.txt, trace.txt, cycles.txt, leds.txt ,display.txt ,diskout.txt

בתוכנית אנו משתמשים במבנה נתונים חדש בשם Command:

```
typedef struct cmd {
	char inst[9];//contains the line as String
	int opcode;
	int rd;
	int rs;
	int rt;
	int imm;
}Command;
```

מבנה נתונים זה מכיל את השורה שמתקבלת בmemin.txt כמחרוזת, וכן את ערכי המספרים המתאימים לכל תו בשורה.

בתחילת התוכנית נגדיר את המשתנים הגלובליים הבאים:

```
#define SIZE 4096
```

זהו גודל הזיכרון המירבי

```
#define SIZE_OF_DISK 16384
```

זהו מספר השורות המירבי שקיים בדיסק. גודלו של הדיסק הוא 128 סקטורים כאשר כל סקטור מכיל 128 שורות. בסימולטור אנו מעתיקים את תוכנו של הדיסק למערך, מבצעים מניפולציות על המערך ואז כותבים את תוכנו חזרה לקובץ הפלט diskout.txt.

```
char file_arr[SIZE + 1][9]
```

זהו מערך של מחרוזות אשר יוזן בתוכנו של קובץ הקלט memin.txt, כאשר כל שורה בקובץ תומר למחרוזת במערך. מכיוון שגודלו של memin.txt מוגבל ב SIZE, מערך זה יכיל SIZE מחרוזות במקרה הקצה המירבי.

```
char disk_out_array[SIZE_OF_DISK + 1][9]
```

מערך זה יכיל את תוכנו של הדיסק הקשיח. כפי שצוין, בתחילת התוכנית נעתיק את תוכנו של diskin.txt למערך, ולאחר ביצוע מניפולציות בהתאם לתוכנית על המערך, נפלוט את תוכנו אל diskout.txt

```
int reg_arr[16]
```

מערך זה ימדל את הרגיסטרים של המעבד.

```
static int memin_array_size
```

באמצעות משתנה זה נשמור את גודלו המעשי של memin.txt, כלומר ימנה את מספר השורות בקובץ.

```
static int pc = 0
```

משתנה זה ימדל את ערך הpc שהמעבד מבצע בכל מחזור.

```
static int count_inst
```

משתנה זה יספור את כמות ההוראות שביצעה התוכנית.

```
static int is_irq1_run
```

משתנה זה ישמש כסמן אשר בודק האם מתבצעת פסיקה מסוג 1.

```
static int count_1024
```

במידה והתקבלה הוראת כתיבה/קריאה לדיסק הקשיח, נספור את כמות מחזורי השעון שעברו מרגע קבלת הפקודה באמצעות משתנה זה.

```
char IOregister[18][9]
```

מערך זה ימדל את רגיסטרי החומרה.

```
static int irq
```

משתנה זה יכיל את מצב הפעולה של הפסיקות.

```
int ready_to_irq
```

משתנה זה יסמן האם הסימולטור פנוי לקבל פסיקה.

```
int irq2_interrupt_pc[SIZE]
```

מערך זה יכיל את כל הpc בהם התקבלה פסיקה מסוג 2 מקובץ הקלט irq2in.

```
int irq2_current_index
```

משתנה זה ישמור את האינדקס של הפסיקה מסוג 2 הנוכחית. כלומר, במערך irq2_interrupt_pc אנו נזין את ערכי הpc המתקבלים מקובץ הקלט, ובאמצעות המשתנה irq2_current_index נזכור את האינדקס הנוכחי במערך. ברגע שנגיע למחזור השעון שבו מתקבלת פסיקה מirq2in, נעלה את האינדקס הנוכחי ב1.

בסימולטור נצהיר על כל הפונקציות הקיימות בתוכנית לצורך בהירות וקימפול נוח.

הסברים על הפונקציות מופיעים בהמשך המסמך.

```
void read_Data_from_irq2in(FILE * irq2in);

void count_to_1024();

void diskhandle(char diskout[][9]);

char * slice_str(char str[], int start, int end);

int HexCharToInt(char h);

void Int_to_Hex8(int dec_num, char hex_num[9]);

int HexToInt2sComp(char * h);
```

```
void FillArray(FILE * memin);

void FillArrayOfdiskout(FILE * diskin);

void RegItOut(FILE * pregout);

void MemItOut(FILE * pmemout);

void DiskItOut(FILE * diskout);

void TraceIt(Command * com, FILE * ptrace);

void TimerHandle();

void irq_status_check();


void hwregtrace(FILE * phwregtrace, int rw, int reg_num);

void BuildCommand(char * command_line, Command * com);

void leds(FILE * pleds);

void display(FILE * pdisplay);

void clk_counter();

void Perform(Command * com, FILE * ptrace, FILE * pcycles, FILE * pmemout, FILE
* pregout, FILE * pleds, FILE * pdiskout, FILE * pdisplay, FILE * phwregtrace, FILE *
pdiskin);

void InstByLine(FILE * ptrace, FILE * pcycles, FILE * pmemout, FILE * pregout, FILE
* pleds, FILE * pdiskout, FILE * pdisplay, FILE * phwregtrace, FILE * pdiskin);
```

<u>אופן הפעולה של הסימולטור:</u>

פונקציית הMain:

הסימולטור כאמור מקבל באמצעות שורת הפקודה קבצי פלט וקלט כמתואר מעלה. תוכנית המain אחראית לפתוח את הקבצים הקיימים וליצור את הקבצים החדשים, ולבדוק שפעולות אלו התבצעו בצורה תקינה. במידה והתוכנית נכשלה בפתיחה או יצירה של הקבצים הדרושים, התוכנית תיסגר עם סטטוס 1 שמסמן על כשלון בריצת התוכנית.

לאחר מכן תתבצע קריאה לפונקציה FillArrayOfdiskout המקבלת את המצביע לקובץ diskin. פונקציה זו תעתיק את תוכנו של הקובץ diskin.txt למערך disk_out_array.

נקרא לפונקציה read_Data_from_irq2in אשר מקבלת מצביעה לקובץ irq2in.txt ותפקידה כאמור להעתיק את תוכנו של הקובץ לתוך מערך מסוג integer בשם irq2_interrupt_pc. נשמור מחזור השעון של הפסיקה הראשונה באינדקס 0 של המערך, את הפסיקה הבאה באינדקס 1 וכך הלאה. בכל ביצוע הוראה נבדוק האם במחזור השעון הנוכחי מתקבלת פסיקה כזאת, ונפעל בהתאם.

נקרא לפונקציה FillArray אשר מקבלת מצביע לקובץ בשם memin.txt ומעתיקה את תוכנו של הקובץ לתוך המערך file_arr. הפונקציה בנוסף מרפדת באפסים את המערך במקומות בהם memin.txt היה ריק. בסיום הפונקציה, נסגור את הקובץ.

נקרא לפונקציה InstByLine אשר מקבלת מצביע לקבצים הבאים:

. trace, cycles, memout regout,leds,diskout,display,hwregtrace,diskin

ותפקידה לבצע את הפקודות שורה אחר שורה ועליה מפורט מטה.

כאשר נחזור מהקריאה לInstByLine חזרה לmain, למעשה כל התוכנית תתבצע ונותר לנו רק לדאוג לייצר את קבצי הפלט המתאימים, לסגור את קבצי התוכנית ולסיים את התוכנית בסטטוס 0.

נכתוב לקובץ cycles את מספר מחזורי השעון שרצה התוכנית.

נקרא לפונקציה DiskItOut אשר מקבלת מצביע לקובץ diskout ומעתיקה את תוכנו של המערך הממדל את הדיסק הקשיח לקובץ.

נקרא לפונקציה RegItOut אשר מקבלת מצביע לקובץ regout ומעתיקה את תוכן הרגיסטרים בסיום הריצה לקובץ.

נקרא לפונקציה MemItOut אשר מקבלת מצביע לקובץ memout וכותבת את תוכן הזיכרון בסיום הריצה לקובץ.

ולסיום נסגור את כל הקבצים.

:InstByLine

תפקידה של הפונקציה הוא לוודא שכל פקודה נעשית בזמן המתאים לה בקוד, החל מ pc=0 ואז לסיום התוכנית. ראשית, הפונקציה יוצרת משתנה חדש מסוג Command, אליו נזין בכל מחזור שעון את הפקודה המתאימה לו. כל איטרציה בלולאה מסמלת ביצוע פקודה חדשה. בכל איטרציה נקרא לפונקציה BuildCommand אשר מקבלת את הפקודה הרלוונטית בהתאם לpc, וכן מקבלת את curr_com הנוכחי, ודואגת להזין אותו בנתונים המתאימים לפקודה לפי הpc המתאים. לאחר שהמשתנה curr_com מכיל את ההוראה המתאימה, נקרא לפונקציה Perform. פונקציה זו מוציאה לפועל למעשה את ההוראה ועליה מפורט בהמשך. לאחר שההוראה מתבצעת באמצעות הPerform , נקרא לפונקציה irq_status_check אשר בודקת האם ההוראה שבוצעה בperform שינתה את אחד מהרגיסטרים המודיעים על פסיקה. במידה והתבצעה פסיקה, הפונקציה משנה את הרגיסטרים המתאימים וכן מעלה את irq ל1. לאחר מכן נבדוק האם התקבלה פסיקה וכן האם היינו מוכנים לקבל פסיקה. במידה וכן, נשנה את הpc לערך המתאים בהתאם לפסיקה, נשמור את הpc הקודם ברגיסטר המתאים ונוריד את הסיגנל ready_to_irq ל0.

:Perform

בשל חשיבותה של הפונקציה בתוכנית, נציג שוב את החתימה שלה:

```
void Perform(Command * com, FILE * ptrace, FILE * pcycles, FILE * pmemout, FILE
* pregout , FILE * pleds, FILE * pdiskout, FILE * pdisplay , FILE * phwregtrace, FILE
*pdiskin)
```

הפונקציה מקבלת את ההוראה שצריך לבצע, וכן מצביע לכל הקבצים המצויינים מעלה, שכן במידה ונקבל פקודה מסוג halt נצטרך לסגור את כל הקבצים לפני סגירת התוכנית.

ראשית, נקרא לפונקציה TraceIt אשר מקבלת את ההוראה וכן מצביע לקובץ trace.txt וכותבת את תוכנה של ההוראה הנוכחית לקובץ לפי הפורמט המבוקש. לאחר מכן ניכנס לswitch אשר מקבל את שדה הopcode של הפקודה, ומבצע את הפקודה המבוקשת. רוב הפקודות טריוויאליות ומבצעות את הדרוש על פי התוכנית, ולכן נציין רק את הפקודות בעלות הקשורות במימוש שלנו.

הפקודה out: בפקודה זו אנו כותבים לרגיסטר חומרה, ולכן ביצוע הכתיבה נקרא בנוסף לפונקציה hwregtrace אשר תפקידה לכתוב לקובץ phwregtrace לפי הפורמט המבוקש, ולאחר מכן נבדוק לאיזה רגיסטר חומרה כתבנו. כתיבה לרגיסטר leds,display תגרור קריאה לפונקציה המתאימה שתכתוב לקובץ את הפלט המתאים. לעומת זאת, כתיבה לרגיסטר diskcmd תיגרור קריאה לפונקציה diskhandle עליה נרחיב בהמשך.

הפקודה halt: פקודה זו מסיימת את התוכנית, ולכן נדאג לבצע את כל הפעולות הדרושות לסיום התוכנית עליהן נרחיב בהמשך ההסבר על הmain.

כאשר נצא מהswitch, נדאג לבצע את שגרת ההוראות הבאות:

קריאה לפונקציה clk_counter, אשר דואגת להעלות את השעון וממששת שעון מחזורי.

קריאה לפונקציה count_to_1024, אשר מונה 1024 מחזורי שעון מהרגע שהתקבלה פקודת קריאה או כתיבה מהדיסק הקשיח. ברגע שהמנייה מגיעה ל1024, הפונקציה דואגת לשנות את הרגיסטרים המתאימים 14,17,4, לאפס את המונה וכן מבצעת השמה של is_irq1_run ל0. בסיום הפונקציה, במידה ולא התקבלה פקודת halt אשר סוגרת את התוכנית, הפונקציה תחזור לInstByLine .

הפונקציה diskhandle:

הפונקציה בודקת ראשית האם הדיסק פנוי לפעולה של קריאה/כתיבה, ואם לא – פעולת הפונקציה מסתיימת. כאשר הדיסק פנוי לפעולת קריאה וכתיבה, נגדיר את משתנה הsector אשר יחושב לפי ערכו של רגיסטר חומרה 15 (register sector) מוכפל בגודלו של כל סקטור – 128. לאחר מכן נבדוק את רגיסטר 14 כדי להחליט האם התקבלה פעולת קריאה, כתיבה, או שהרגיסטר במצב No-command. בפעולת קריאה נעתיק את ערכו של הדיסק בסקטור המתאים לזכרון, ובפעולת כתיבה נבצע את הפעולה ההפוכה.