פרויקט מבנה המחשב

Assembler:

תוכנית האסמבלר כוללת 3 מערכים אשר יכילו את פקודות האסמבלי, התוויות ומערך הזיכרון כמו גם מונה מתאים עבור כל מערך.

```
static Command CommandDB[ MAX_NUM_OF_COMMANDS ] = { 0 };
static int GlobalPC = 0;
static Lable LableDB[ DB_MAX_NUM_LABLES ] = { 0 };
static int lableIndex = 0;
static Memory MemoryDB[ MAX_NUM_OF_COMMANDS ];
static int memIndex = 0;
```

להלן המבנים שהפרויקט נעזר בהם:

```
typedef struct {
    int PC;
    int Opcode;
    int RD;
    int RS;
    int RT;
    int RM;
    char Imm1[ BUFFER_MAX_SIZE ];
    char Imm2[ BUFFER_MAX_SIZE ];
    char Lable[ BUFFER_MAX_SIZE ];
    char Note[ BUFFER_MAX_SIZE ];
} Command;

typedef struct {
    char Lable_name[ BUFFER_MAX_SIZE ];
    int PC;
    int is_created;
} Lable;

typedef struct {
    int location;
    char value[ BUFFER_MAX_SIZE ];
} Memory;
```

האסמבלר קורא את קובץ תוכנית האסמבלי שורה שורה ומבצע את הפעולות הבאות:

1. בדיקה האם השורה ריקה או כוללת הערה בלבד – במידה וכן ממשיך לשורה הבאה.
2. הפרדת השורה לשני חלקים: העברה מול תווית+פקודה. את ההערה יזין לשדה המתאים וימשיך עם פרסור התווית+פקודה.
3. בדיקה האם השורה מכילה תווית, פקודת זיכרון ו/או פקודה.
   לפי התוצאה האסמבלר יפרסר בצורה המתאימה את השורה ויזינה למבנה הפקודה, למערך הזיכרון ולמערך התוויות (תוך עדכון שורת הפקודה אליה יש לקפוץ) בהתאם.
   במידה וקיים שדה IMM או תווית נשאירם בשלב זה את התוכן כמחרוזת ונטפל בכך בשלב הבא.
4. בסיום המעבר הראשון על הקובץ נעבור על מערך הפקודות שיצרנו במהלכו ונמיר כל ערך IMM בהתאם לכתוב בפקודה (המרה למספר או בדיקה במערך התוויות והמרה לשורה המתאימה לפי הרשום המערך) ונכתוב את קבצי הפלט של האסמבלר dmemin.txt, imemin.txt.

simulator:

שני מערכים שיכילו את ערכי הרגיסטרים מקבצי הקלט ומערך שיכיל את ערכי הזיכרון.

```
int registers_values[NUM_OF_REGISTERS] = { 0 };
uint32_t io_registers_values[NUM_OF_IO_REGISTERS] = { 0 };
uint32_t memory[MEM_SIZE] = { 0 };
```

struct של command שיכיל את כל רכיבי הפקודה

```
typedef struct {
uint32_t PC;
char INST[CMD_LENGTH_HEX + 1];
uint32_t Opcode;
uint32_t RD;
uint32_t RS;
uint32_t RT;
uint32_t RM;
int Imm1;
int Imm2;
} Command;
```

מערך שיכיל את כל הפקודות שנקראות מקובץ הקלט

```
Command* commands[MAX_NUM_OF_COMMANDS] = { NULL };
```

קריאה של קבצי הקלט

```
void read_imemin_file()
void read_dmemin_file()
```

פרסור של שורת קלט בודדת מקובץ הרגיסטרים, יצירת struct של command כנ"ל והכנסה למערך commands

```
void parse_cmd_line(char *line, int local_pc)
```

בדיקה האם שורה ריקה או מכילה הערה, משמש עבור פרסור קבצי הקלט

```
bool isLineEmptyOrNoteOnly( char *line )
```

מעבר על מערך הפקודות וביצוע אחת כל מחזור שעון, כמו כן בדיקה של אוגרי הפסיקות.

```
void simulator()
```

ביצוע הפקודה אשר מתקבלת מהסימולטור בכל מחזור שעון

```
bool call_action(Command *cmd)
```

Disk:

מערך דו ממדי אשר יכיל את ערכי הקלט של הדיסק, כאשר כל מערך הוא סקטור

```
uint32_t diskMemory[SECTOR_NUMBER][SECTOR_SIZE];
```

אתחול המערך הנ"ל לפי קובץ הקלט של הדיסק

```
void initDisk()
```

בדיקה כל מחזור שעון האם יש בקשת גישה לדיסק וכן טיפול בזמן הריצה של הדיסק

```
void diskHandler()
```

קריאה/כתיבה מהדיסק בהתאם לערך שנשמר בdiskcmd

```
void readFromDisk()
void writeToDisk()
```

interrupt:

מערך המכיל את זמני המחזור בהן יש קריאה לפסיקה 2

```
int* irq2Arr;
```

אתחול המערך הנ"ל

```
void initIrq2()
```

משתנה המכיל האם המעבד נמצא כרגע בטיפול בפסיקה

```
int isInterrupt;
```

חזרה מפסיקה, עדכון המשתנה הנ"ל והpc

```
void return_from_interrupt()
```

בדיקה בכל מחזור שעון האם יש פסיקה חדשה, ואם המעבד לא בזמן טיפול בפסיקה אחרת עדכון אוגר החזרה מפסיקה והpc

```
int interruptHandler()
```

טיפול בפסיקה הטיימר כל מחזור שעון – קידום שעון אם מופעל ועדכון האוגרים אם הגיע למקסימום

```
void updateIrq0()
```

טיפול בפסיקה 2 כל מחזור שעון – בדיקה האם יש פסיקה במחזור שעון הנוכחי ועדכון סטטוס אם כן

```
void updateIrq2()
```

Files:

פתיחת קבצים

```
void set_FD_context( char *argv[] )
```

כתיבה לקובץ hwregtrace בכל מחזור שעון בו נקראת פקודת In/out

```
void add_to_hwregtrace_file(Command *cmd)
```

כתיבה לקובץ trace בכל מחזור שעון

```
void add_to_trace_file(Command *cmd)
```

כתיבה לקבצי הפלט

```
write_cycles_file();
write_regout_file();
write_dmemout_file();
write_diskout_file();
write_monitor_file();
```

סגירת קבצים

```
void close_FD_context()
```

assembly

disktest – התחלה מסקטור 7, קריאה שלו לdiskbuffer ואז כתיבה שלו לסקטור 8, ואז קריאה של סקטור 6 וכתיבה שלו ל7 (שכבר הועתק) וכך הלאה. בין כל בקשת גישה יש קפיצה ללולאת wait שמחכה לישון של סטטוס הדיסק.

Mulmat- שתי לולאות for מקוננות, כאשר הראשונה רצה על השורות והשנייה על העמודות, בלולאה הפנימית מתבצע חישוב ערך מטריצת הפלט על ידי כפל וסכימה של הערכים המתאימים.

Binom – קריאה רקורסיבית לסכום פונקציות עם ערכים קטנים יותר עד הגעה לתנאי עצירה (לפי הנוסחא)

Circle – נקרא את ערך הרדיוס מהזיכרון ונחשב את ערכו בריבוע. נעבור על הפיקסלים במוניטור בלולאה מקוננת ועבור כל פיקסל נחשב את המרחק בריבוע שלו ממרכז המוניטור. במידה והמרחק קטן או שווה מריבוע הרדיוס שקראנו מהזיכרון נשנה את ערך הפיקסל ללבן, אחרת נשאיר אותו עם ערכו הדיפולטי, שחור.