ISA Project | Computer organization course

דן קומיסרצ'יק 316381946

דור אנדר 205541683

טל כהן 205918568

בקובץ זה נפרט על עיקרון הפעולה של הפרויקט ונחלקים השונים אותם מימשנו.

**האסמבלר**

האסמבלר בנוי מתוכנית בשפת C אשר מקבלת קטע קוד באסמבלי (קובץ הקלט) ומדפיסה לקובץ פלט אשר נקרא memin. קובץ הפלט הינו הקידוד הבינארי של קוסץ האסמבלי לשפת המעבד כפי שהוגדר בפרויקט.  
אפרט בשלבים את הפעולות שהאסמבלר עושה על מנת להדפיס קובץ פלט תקין:

**שלב ראשון**

תחילה קראנו את קובץ הקלט לפי שורותיו על ידי פונקציה בשם – set\_lines. פונקציה זו מקבל מערך כגודל הזיכרון והכנסנו כל שורה בקובץ האמסבלי (גם אם היא ריקה) אל המערך שלנו, בנוסף הפונקציה נשלחת אל פונקציה אחרת אשר מוחקת את ההערות שמופיעות לאחר ה - #. לאחר מכן על מנת להתעלם מן רווחים בתחילת השורה שלחו את המערך לפונקציה בשם delete\_spaces\_start\_lines על מנת להתעלם מן הרווחים הלא רצויים.  
לאחר מכן מחקנו את כל השורות הריקות מן המערך על ידי פונקציה בשם – delete\_empty\_lines.  
אציין כי בשלב זה טיפלנו רק בהוראות אשר אינן .word אלא הוראות רגילות.

**שלב שני**

בשלב זה נרצה לעבור על הקובץ ולהגדיר את התוויות של קובץ האסמבלי לתוך מערך נפרד. פונקציה בשם set\_labels עושה זאת על ידי כך שהיא מקבלת את מערך השורות ובנוסף מערך ריק כגודל הזיכרון. כאשר מופיעה תווית בשורה מסוימת במערך אנו נכניס את התווית אל המערך באינדקס השווה לשורה שמופיעה בה התווית. נוכל לזהות שורה באופן חד ערכי על ידי ':' שכן מחקנו את ההערות ואנחנו מניחים קלט תקין.  
לאחר שלב זה מחקנו את התוויות מן המערך של השורות וכן מחקנו שורות ריקות.

**שלב שלישי**

בשלב זה נרצה לטפל בהוראות .word על ידי מערך נוסף בגודל הזיכרון אשר אינדקס המערך מייצג את הכתובת והתוכן בתא במערך מייצג את המידע שנרצה לכתוב. הפונקציה הנ"ל מקבלת את המערך של השורות ומערך ריק שהמייצג בסופו של דבר את הוראות ה - .word

**שלב רביעי**

בשלב זה נרצה לעבור על המערך של השורות וכן על מערך התוויות על מנת להבין איזה ערך מסמל התווית. נזכור כי יש לנו הוראות מסוג I-type ו ו R-type אשר לא תופסות את אותו המקום בזיכרון שלנו. לפיכך קראנו לפונקציה set\_jump\_line\_labels אשר תקפידה לקבל מערך של שורות ותוויות וליצור מערך בגודל הזיכרון אשר האינדקס בו מסמל את השורה שבה התווית נמצאת והתוכן של התא מסמל את השורה שאליה נרצה לקפוץ בקובץ ה- memin.

**שלב חמישי**

על מנת לשלוט בכל המידע שיש ברשותנו בצורה מסודרת הגדרנו מבנה באופן הבא:

typedef struct Instruction

{

char opcode[8];//opcode number

char rd[4]; // rd register

char rs[4]; //rs register

char rt[4]; //rt register

char imm[20]; //immediate

int rtype\_or\_itype; //decide if rtype of itype

int word\_or\_regular; // decide if regular or word instruction

int adress; //if word instruction - the adress we want to write to

int data; //if word instruction - the data we want to write

}inst;

בפונקצית ה – main יצרנו מערך מסוג המבנה הנ"ל בגודל הזיכרון. בעזרת פונקציה בשם set\_instructions המרנו כל שורה בקובץ האסמבלי (מערך השורות, מערך התוויות ומערך הפסודו-הוראות) להוראה חוקית לפי המבנה. במעבר על השורות הבחנו בין הוראות מסוג I-type/R-type, הבחנו בפקודות של I-type בין תוויות, מספר בהקסא ומספר דצימלי.   
נציין כי נעזרנו בפונקציות עזר אשר מתועדות בצורה מפורטת בקוד.

**שלב שישי**

בשלב זה נכתוב את הנתונים ממערך ההוראות שלנו אל קובץ הפלט memin בעזרת פונקציה בשם write\_memin.  
לאחר קריאה לפונקציה זו נזכור כי ביצענו הקצאות דינמיות על מנת להגדיר היטב את המערכים שלנו ולכן נשלח את כולם אל פונקציה בשם release\_all אשר משחררת הקצאות דינמיות מכל המערכים.

**סימולטור Simulator –**

הסימולטור מייצג באופן אבסטרקטי את המעבד ונכתב בשפת C.

הסימולטור מקבל 3 קבצי קלט:

* Memin.txt – קובץ שמיוצר ע"י האסמבלר ומייצג את תמונת הזיכרון הראשית של התוכנה. הקובץ מיוצר ע"י האסמבלר אופן אוטומטי, תמונת הזיכרון בנויה משורות עומק תמונת הזיכרון יהיה לכל היותר 4096 שורות.

כל שורה ברוחב 20 ביטים בייצוג HEX לפי הסדר בהנחיות לפרויקט ותייצג פקודה או דאטה (במידה ושורה לפני מתבצעת פקודת Itype).

* Diskin.txt - קובץ זה מייצג את תמונת הזיכרון של הדיסק (מצב הזיכרון הנוכחי שלי הדיסק) ויגיע עם שאר התוכנית לסימולטור. (פורמט הקובץ זהה לmemin פרט לעומק הזיכרון שיהיה 128 סקטורים \* 128 רוחב סקטור)
* Irq2in - בקובץ זה שורה אחר שורה בייצוג מספרי רשומים זמני המחזור עבורם נדרשים לבצע פסיקת irq2.

תפקידו של הסימולטור באופן כללי הוא לדמות את פעילותו של מעבד חומרתי ואת שלל הפעולות שיבצע למשל פיענוח פקודות, ביצוע הפעולות באופן מחזורי והוצאת קלטים (I/O) למשתמש. את פעילות הסימולטור נחלק למספר חלקים עיקריים:

* **Fetch – decode** – ברצף פעולות זה הפונ' הראשית :

void exe\_inst(int32\_t mem[], int32\_t R[], int inst\_num, IO\_HW\* hardware, FILE\* trace)

תיגש בכל מחזור שעון חדש לתמונות הזיכרון תנתח את הפקודה הנוכחית, במידה ופקודת itype גם תיגש לדאטה בשורה הבאה. בשלב זה מתבצע ניתוח של הפקודה ובעזרת שימוש בstruct שמכיל את המידע המלא על כל פקודה תבצע ניתוח מלא שיעבור לפונ' הביצוע.

* **Execute** – בשלב זה ההוראה נותחה וכל המידע הרלוונטי שמור וכעת תעבור לפונ' הביצוע:

int exe\_opcode(command com, int32\_t mem[], int32\_t R[], int pc, IO\_HW \* hardware)

הפונ' אחראית לבצע את הפקודה (פקודה מרשימת הפקודות הקיימות) המתאימה בעזרת הרגיסטרים המוקצים לתוכנית על סמך הopcode הנתון ונותח בשלב הקודם. בסיום הפעולה נעדכן לפי סוג הפעולה את pc ונקדם מחזורי שעון לפי סוג הפעולה שהתקבלה.

* **Interruptions handler** – חלק זה של הסימולטור אחראי על ניהול שגרת הפסיקות, בכל מחזור שעון נבדוק האם התקבלה פסיקה ולאחר מכן תיבדק האפשרות האם יש לבצע פסיקה לפי הפונ': int intterupted(int pc, IO\_HW\* hardware)

במידה וכן התוכנית "תקפוץ" לpc שחולץ מרגיסטר החומרה ומייצג את מיקום תחילת ביצוע שגרת הפסיקה.

* **I\O Hardware** – כל המידע החומרתי (פסיקות, דיסק, טיימר, נורות לד, מוניטור וכו') נמצא במבנה IO\_HW . מבנה זה ישמש אותנו להתנהלות עם החומרה שקשורה לפעילות קלט/פלט ובקרת פסיקות.

הסימולטור מוציא מספר קבצי קלט:

* Memout.txt – תמונת הזיכרון בתום ריצת התוכנית (בפורמט זהה לשל תמונת הזיכרון בתחילת התוכנית).
* Regout.txt – תמונת הרגיסטרים המוקצים לתוכנית בתום ריצת התוכנית (מופיע כל רגיסטר ותוכנו)
* Trace.txt – קובץ המתאר בכל מחזור שעון את סוג הפעולה שהתבצעה, ערך הpc בעת הביצוע וערכי הרגיסטרים שמוקצים לתוכנה.
* Hwregtrace.txt – קובץ המתעד בכל מחזור שעון ביצוע של פעולת כתיבה/קריאה לרגיסטר חומרה ומה סוג הפעולה שהתבצעה וגם המידע הנכתב/נקרא.
* Cycles.txt – קובץ המתאר את מספר מחזורי השעון בתום הריצה
* Leds.txt – קובץ המתאר את מערך נורות הלד והשינוי במהלך הריצה.
* Display7seg.txt – קובץ המתאר שינוי בתוכנית במהלך הריצה במערך 7 segment display.
* Diskout.txt – תמונת הזיכרון של הדיסק בתום ריצת התוכנית ( בפורמט זהה לתמונת הזיכרון בתחילת התוכנית)
* Monitor.txt – קובץ המתאר את תוכן המוניטור בתום ריצת התוכנית.
* Monitor.yuv – קובץ בינארי המתאר את תוכן המוניטור בתום ריצת התוכנית ומשמש לתוכנה שתפענח את הקובץ ותיתן תמונה גרפית של תוכן המוניטור.

חשוב להזכיר שקיימות פונקציות עזר רבות נוספות בתוכנית שמשמות לפעולות חישוב ופעולות קריאה וכתיבה לקבצים. הפונקציות כולן מתועדות בתוכנת הסימולטור עם הוראות מפורטות שמסבירות את משמעותן. בתיעוד זה בחרנו להתמקד בצורה הכללית והעקרונות לפיהן עובד הסימולטור וכיצד בעצם מדמה מעבד חומרתי.

**תכניות בדיקה (אסמבלי):**

**Sort:**

תחילה אתחלנו מערך של מספרים לכתובות.

לאחר מכן שמרנו את הכתובת של תחילת המערך ושל הסוף.

Loop1- טוענים מהזיכרון את המספר אותו אנו רוצים להשוות עם המספרים הקודמים.

בודקים שלא סיימנו את המערך.

Loop2- רצים על המספרים במערך ובודקים אם הם גדולים או קטנים מהמספר שמשווים עכשיו( עד למקום של המספר העכשווי). במידה והם גדולים ממנו הולכים לפונקציית swap, במידה ועברנו את המקום במערך אותו אנו רוצים להשוות הולכים לאתחול before\_loop1.

Swap- מחליפה את מיקום של המספרים.

**Binom:**

בהתחלה מאתחלים שני מספרים n,k. לאחר מכן בmain זה המקום בו טוענים את הערכים כארגומנטים לפונקציה המרכזית binom. זוהי פונקציה רקורסיבית אשר בתחילה שומרים במחסנית את הארגומנטים שלה ואת כתובת החזרה. לאחר מכן בודקים את תנאי העצירה לרקורסיה ואם לא קוראים ממנה לקריאות רקורסיביות. במידה ונתקלים בתנאי עצירה קופצים לreturn1 שבה מחזירים 1 על מנת לסכום אותו. במידה וסיימנו את הרקורסיה קופצים לreturn ושם מחזירים בסופו של דבר חוזרים לmain ושומרים את הערך.

**Triangle:**

אלגוריתם הצביעה עובד כך: בודקים מי יותר גדול מבין AB, BC. במידה וAB גדול יותר נמלא עמודה עמודה משמאל לימין ומלמעלה למטה. במידה וBC גדול יותר נמלא שורה שורה מלמעלה למטה מימין לשמאל. אסביר את החישובים למקרה שך AB>BC. מחשבים FLOOR(AB/BC) ואת הערך AB/(FLOOR(AB/BC)). כאשר הערך השני ייצג לנו את ההפרש בין שתי עמודות עוקבות. לאחר מכן ממלאים את AB. אחר כך ממשיכים למילוי המשולש כאשר כל פעם עוברים עמודה אחת ימינה וממלאים מלמעלה למטה כאשר הלמעלה מחושב עם הערך ממקודם.

**:DISKTEST**

אלגוריתם הסכימה עובד כך: בכל פעם טוענים סקטור שלם לכתובת 3000-3127. שומרים את המילה הi שלו וטוענים את הסקטור הבא ככה שמונה פעם כאשר כל פעם מוסיפים את המילה הi לסכימה. אחרי שחיברנו את המילה הi בכל שמונת הסקטורים הראשונים אנו שומרים את זה במקום המתאים בכתובת 3128-3255. אחרי שסיימנו את כל ה128 מילים טוענים את הערך לסקטור 8 בדיסק. כלומר אנחנו עושים 128 לולאות חיצוניות כאשר בכל אחת מהן יש 8 לולאות פנימיות. כמובן לפני כל קריאה מהדיסק ולפני הכתיבה שלנו אנחנו בודקים שהדיסק פנוי.