לירן גלמידי 208536474
ארז ורדניקוב 316275478
שלומי קוזרי 315977157
סיון פרדה 209343649

מבנה המחשב- דוקומנטציה לפרויקט ISA

רקע כללי:

בפרויקט זה נכתוב תוכניות בשפת אסמבלי עבור מעבד RISC בשם SIMP על ידי מימוש אסמבלר וסימולטור בשפת סי. כמו כן, נכתוב מספר תוכניות בדיקה באסמבלי ונראה את נכונות הקוד על ידי הרצת מספר דוגמאות.

האסמבלר

רקע כללי:

האסמבלר נכתב בשפת סי, ומתרגם את תוכניות האסמבלי שכתובות בטקסט בשפת אסמבלי, לשפת המכונה בהצגה בבסיס 16 .
קובץ **הקלט** asm.program מכיל את תוכנית האסמבלי, וקובץ **הפלט** memin.txt מכיל את תמונת הזיכרון. קובץ הפלט של האסמבלר משמש אח״כ כקובץ הקלט של הסימולטור.
כל שורת קוד בקובץ האסמבלי מכילה את כל 5 הפרמטרים בקידוד ההוראה אשר יקודדו לשפת מכונה באורך 8 ביטים , כאשר הפרמטר הראשון הינו הopcode, ושאר הפרמטרים מופרדים ע״י סימני פסיק. לאחר הפרמטר האחרון מותר להוסיף את הסימן # והערה מצד ימין.

כדי לתמוך ב-labels האסמבלר מבצע שני מעברים על הקוד.
במעבר הראשון זוכרים את הכתובות של כל ה-labels ובמעבר השני בכל מקום שהיה שימוש ב-labels בשדה ה- immediate מחליפים אותו בכתובת ה-label בפועל כפי שחושב במעבר הראשון.

בנוסף להוראות הקוד, האסמבלר תומך בהוראה נוספת המאפשרת לקבוע תוכן של מילה 32 ישירות בזיכרון. הוראה זו מאפשרת לקבוע דאטא בקובץ תמונת הזיכרון word address data. .
כאשר address הינו כתובת המילה ו-data תוכנה. כל אחד משני השדות יכול להיות בדצימלי, או הקסאדצימלי בתוספת 0X .

בנוסף על האסמבלר לתמוך בתרגום נכון עבור 3 אפשרויות לשדה ה immediate בתוכנית האסמבלי :

- ניתן לשים שם מספר דצימלי, חיובי או שלילי.
- ניתן לשים מספר הקסאדצימלי שמתחיל ב x0 -ואז ספרות הקסאדצימליות.
- ניתן לשים שם סימבולי שמתחיל באות. במקרה זה הכוונה ל label -כאשר label מוגדר בקוד ע״י אותו השם ותוספת נקודתיים.

הנחות נוספות :

1. ניתן להניח שאורך השורה המקסימאלי בקבצי הקלט הוא 500
2. ניתן להניח שאורך ה label -המקסימאלי הוא 50
3. פורמט ה label -מתחיל באות, ואח"כ כל האותיות והמספרים מותרים.
4. צריך להתעלם מ whitespaces -כגון רווח או טאב. מותר שיהיו מספר רווחים או טאבים ועדיין הקלט נחשב תקין.
5. יש לתמוך בספרות הקסאדצימליות גם ב case lower -וגם.case upper
6. מרחב הכתובות במעבד SIMP הינם ברוחב 9 סיביות בלבד (זיכרון 512 מילים.)
7. במידה ומספר השורות בקובץ קטן מ- 512, ההנחה הינה ששאר הזיכרון מעל הכתובת האחרונה שאותחלה בקובץ, מאופס.

הסבר על החלקים בקוד האסמבלר:

הגדרת גדלים קבועים :גודל הזיכרון ,אורך לייבל מקסימאלי ואורך שורה מקסימלי בהתאם להנחות נוספות ברקע הכללי :

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define MAX_LINE_LENGTH  500
#define MAX_LABEL_LENGTH 50
#define MEMORY_SPACE 512
```

הגדרות מבנה נתונים הבאים :

1. מבנה נתונים בשם line המבטא שורה בקוד האסמבלי -אשר מוגדרת להיות מחרוזת באורך MAX_LINE_LENGTH+1 (כולל את הסיומת '\0') :

```
typedef char line[MAX_LINE_LENGTH + 1];
```

2. מבנה נתונים בשם label המכיל את שם הלייבל -מחרוזת באורך MAX_LABEL_LENGTH+1 ואת מיקומו בקוד :

```
//Data structure of labels.
typedef struct {
    char label_name[MAX_LABEL_LENGTH + 1];
    int label_address;
} label;
```

3. מבנה נתונים בשם line_fields עבור שורת הוראה המכיל את כל שדות שורת פקודה בהתאם לפקודה בקוד אסמבלי:

```
//Data structure of instruction which includes the fields opcode ,rd etc...
typedef struct {
    char* label;
    char* opcode;
    char* rd;
    char* rs;
    char* rt;
    char* immediate;
} line_fields;
```

4. מבנה נתונים בשם line_type המכיל קבועים המגדירים את סוג השורה בהתאם להנחיות הפרוייקט,על מנת לטפל בה בהתאם:

```
//Define line type.
typedef enum {
    WORD,
    LABEL,
    COMMAND,
    LABEL_COMMAND,
    EMPTY_LINE,
} line_type;
```

<u>הגדרות משתנים גלובליים:</u>

המשתנים הגלובליים בהם נעשה שימוש בתוכנית האסמבלר:

```
//global parameteres
label labels[MEMORY_SPACE]; //global array to store the labels and their location.
int labels_amount;
char memory[MEMORY_SPACE][9]; //global array to store the memory
int label_counter = 0;
int enc[3] = { 0 };
char enchex[4] = { '0','0','0' };
char encInstruction[9] = "00000000";
```

1. מערך בגודל הזיכרון המחזיק בכל תא איבר מסוג מבנה הנתונים label.
2. מערך דו מימדי בגודל 512X9 אשר יחזיק את תמונת הזיכרון בסוף הפרוצדורה (memin)
3. מספר שלם המחזיק את כמות הלייבלים שהופיעו בקוד האסמבלי, הן בשורות מסוג LABLE והן בשורות מסוג LABLE_COMMAND .
4. מספר שלם המחזיק את כמות הלייבלים שהופיעו בקוד האסמבלי בשורות מסוג LABLE_COMMAND .
5. מערך בשם enc מסוג int מאותחל לאפסים בגודל 3 אשר מטרתו להמיר כל שדה בשורת ההוראה לייצוג בבסיס 16 .(בכל תא במערך זה יהיה מספר בטווח 1-16).
6. מערך בשם enchex מסוג char מאותחל ל'0' בגודל 4 אשר מטרתו להמיר כל מספר בתא של מערך enc לערך בטווח 1-F.
7. מערך מסוג char בשם encInsruction בגודל 9 אשר מאותחל לאפסים ומטרתו להחזיק בכל פעם שורה שלמה מקודדת לשפת המכונה.

פונקציית אתחול הזיכרון לאפסים (בהתאם להנחה 7 ברקע הכללי) :

```
//initialize_memory to zeros
void initialize_memory(char memory[][9]) {
    for (int i = 0; i < MEMORY_SPACE; i++) {
        strcpy(memory[i], "00000000");
    }
}
```

הכרזות עבור כל הפונקציות בהן יעשה שימוש בתוכנית האסמבלר :

```
//Decleration of all functions in the program.
line_fields parting_lines_to_fields(char* line);
line_type kind_of_line(line_fields field);
void execute(FILE* input_file, FILE* output_file);
void first_pass(FILE* input_file);
int second_pass(FILE* input_file);
int opcode_string_to_num(char* opcode);
int register_string_to_num(char* reg);
int imm_to_num(char* imm);
void encode_num_to_enc_and_enhex(int* enc, char* enchex, int dec);
void encode_instruction(int opcode, int Rd, int Rs, int Rt, int imm, char* encInstruction);
int Hexa_Int_2s(char* hexnum);
int Hex_char_to_int(char h);
char int_to_hex_char(int num);
```

הפונקציות בהן נשתמש :

1) הפונקציה opcode_string_to_num : `int opcode_string_to_num(char* opcode_str)`

פונקציה זו תקבל את שדה ה-opcode בהוראה ותחזיר את הערך המספרי שלו בהתאם לטבלה המצויה בהנחיות הפרויקט למשל עבור הוראה המכילה שדה opcode עם הפקודה jal הפונקציה תחזיר את הערך 13 .

2) הפונקציה register_string_to_num : `int register_string_to_num(char* registr_str)`

פונקציה זו תקבל את שדות הרגיסטרים בהוראה: rd,rs,rt בהוראה ותחזיר את הערך המספרי שלהן בהתאם לטבלה המצויה בהנחיות הפרויקט למשל עבור הוראה שבה שדה הרגיסטר rd הוא $a0 נחזיר את הערך 3 .

3) הפונקציה Hex_char_to_int : `int Hex_char_to_int(char Hex)`

פונקציה זו תקבל char אשר מציג מספר בבסיס 16 (כאשר ניתן להניח כי הקלט תקין) ולכן בטווח '0'-'9' או בטווחים 'a'-'f' , 'A'-'F' וימיר אותה לערך הדצימלי שלה .פונקציה זו תומכת באותיות קטנות וגדולות כפי שנתבקשנו בפרויקט.

4) הפונקציה int_to_hex_char : `char int_to_hex_char(int num)`

הפונקציה ההפוכה לפונקציה הקודמת – פונקציה זו תיקח מספר בטווח 0-15 ותמיר לאות בטווח '0'-'F'.

5) הפונקציה Hexa_Int_2s : `int Hex_char_to_int(char Hex)`

תפקיד הפונקציה הוא להמיר מחרוזת המייצגת מספר בבסיס 16 למספר שלם חיובי\שלילי .

6) הפונקציה parting_lines_to_fields : `line_fields parting_lines_to_fields(char* line)`

פונקציה זו תקבל שורה , אשר תבטא שורה בקוד האסמבלי ותחזיר את מבנה הנתונים line_fields אשר יאכלס בתוכו את הערכים המתאימים ל-opcode,rd וכו' בהתאם להוראה

פונקציה זו מטפלת ברווחים ובהערות #comment בהתאם להנחיות נוספות ברקע הכללי.

7) הפונקציה kind_of_line : `line_type kind_of_line(line_fields field)`

פונקציה זו תקבל את מבנה הנתונים line_fields אשר יאכלס בתוכו את הערכים המתאימים ל-opcode,rd וכו' בהתאם להוראה, ובהתאם לכך תחזיר את אחד מהקבועים במבנה הנתונים line_type.

8) הפונקציה encode_num_to_enc_and_enhex :

```
void encode_num_to_enc_and_enhex(int* enc, char* enchex, int dec)
```

לאחר הקידוד של ה opcode והרגיסטרים וה-imm בשורת ההוראה למספר שלם , נרצה להמיר אותם להצגה ההקסדצימאלית משום שכך הן כתובות בשפת המכונה- לשם כך נועדה פונקציה זו כאשר תחילה נמיר את המספר אל תוך מערך enc :

למשל בהנחה ושדה ה immediate תורגם ל 266 אז פונקציה זו תמלא את מערך enc באופן הבא :

1	0	10

שכן מתקיים :

$$266 = 16^2 \cdot 1 + 16^1 \cdot 0 + 16^0 \cdot 10$$

לאחר מכן יבוצע תרגום של מערך זה למערך enchex באמצעות פונקציות קודמות באופן הבא :

1	0	A

9) הפונקציה encode_instruction :

```
void encode_instruction(int opcode, int Rd, int Rs, int Rt, int imm, char* encInstruction)
```

הפונקציה תשתמש בפונקציה הקודמת לקודד כל אחד מן השדות לייצוג ההקסדצימאלי שלו ותמזג את כולם לכדי שורה שלמה בשפת מכונה . כמו כן, פונקציה זו מבחינה בין שורה מסוג word. שעבורה כתלות בסימן המספר חיובי או שלילי יש לשרשר "00000" או "FFFFFF" .

10) הפונקציה imm_to_num : `int imm_to_num(char* imm)`

הפונקציה תקבל את שדה ה-imm ותתרגם אותה לערכו המספרי, כפי שנכתב ברקע התיאורטי הפונקציה תומכת בכל שלושת האפשרויות בשדה זה.

11) הפונקציה first_pass : `void first_pass(FILE* input_file)`

פונקציה זו תקבל כקלט קובץ (תכנית בדיקה) ותבצע מעבר ראשוני על הקובץ לאתר את כל הלייבלים ולסמן את שמותיהם ומיקומיהם בקובץ.

12) הפונקציה second_pass :

```
int second_pass(FILE* input_file)
```

פונקציה זו תקבל כקלט קובץ (תכנית בדיקה) ותבצע מעבר נוסף על הקובץ בו יתבצע התרגום לשפת מכונה. הפונקציה תחזיר בפועל מספר שלם המבטא כמה שורות קוד תורגמו לשפת מכונה משום שעל פי הנחה 7 ברקע התאורטי במידה ומספר השורות בקובץ קטן מ-512, ההנחה הינה ששאר הזיכרון מעל הכתובת האחרונה שאותחלה בקובץ, מאופס ולכן אין צורך להדפיס שורות אפסים.

13) הפונקציה execute :

```
void execute(FILE* input_file, FILE* output_file)
```

פונקציה זו מקבלת כקלט שני קבצים (קובלט הקלט=תכנית האסמבלי , קובץ אשר יבטא את הפלט memin.txt) הפונקציה תאתחל את הזיכרון ,תבצע מעבר ראשון ושני על תכנית האסמבלי ותכתוב זאת לקובץ הפלט.

14) הפונקציה main :

```
int main(int argc, char* argv[])
```

פונקציה זו תפתח לפעולת קריאה את קובץ האסמבלי ותפתח קובץ לפעולת כתיבה עבור קובץ ה memin.txt, במידה ולא ניתן לפתוח את הקבצים נקבל הודעת שגיאה , אחרת נבצע את הפונקציה execute

סימולטור:

רקע כללי:

תפקיד הסימולטור הינו לסמלץ את לולאת ה-fetch-decode-excuete, משמע תפקידו לקבל את קובץ memin.txt המכיל את הפקודות מהפונקציה רצויה (שנכתבה בשפת אסמבלי) כמחרוזות בבסיס הקסדצימלי ויודע לפענח את המחרוזות האלה לכדי פעולות לביצוע על הרגיסטרים השונים שמהם הוא מורכב.

הקלט אותו מקבל הסימולטור הינו קובץ memin.txt המכיל את שורות הקוד של הפונקציה הנדרשת כשורות של מחרוזות הקסדצימליות.

הפלט אותו מוציא הסימולטור הינו:

1. **Regout.txt**- קובץ המכיל את תכולת הרגיסטרים r2-r15 בסוף הרצת הפונקציה שהגיעה מקובץ הקלט כמספר הקסדצימלי. נבחין כי ערך רגיסטר r0 הינו אפס קבוע והרגיסטר r1 מכיל את ערך ה-immediate.
2. **Cycles.txt**- קובץ המכיל את מספר הפעולות שביצע המעבד על מנת להשלים את הפעולות של הפונקציה שהגיעה מקובץ הקלט.
3. **Trace.txt**- קובץ המכיל את תכולות הרגיסטרים בכל שורה המתבצעת במהלך עבודת הסימולטור **לפני** ביצוע ההוראה. בכל שורה יופיעו כתובת שורת הפקודה (PC), ערך ה-inst שהינו הפקודה המבוצעת בפורמט של בסיס הקסדצימלי, ערך הרגיסטרים של r0 (המכיל את הקבוע 0), r1 (המכיל את ערך ה-immediate) ושאר הרגיסטרים לפני ביצוע הפעולה. הפורמט בו הקובץ נכתב הינו:

   PC INST R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 R15

   כאשר כל ערך מהשורה הנ"ל נכתב כמחרוזת הקסדצימלית באורך של 8 ספרות.
4. **Memout.txt**- קובץ המכיל את תכולת הזיכרון הראשי בסוף ריצת הסימולטור (בדומה לקובץ הקלט memin.txt). הסימולטור כתוב כך, שקובץ זה יכיל 512 שורות בכל מקרה. כאשר במידה ונדרשים פחות שורות לביצוע הפעולה ע"י הסימולטור, בשאר השורות יכתבו אפסים.

הסבר על החלקים בקוד האסמבלר:

הגדרת גדלים קבועים- בתחילת התוכנית נגדיר את הקבועים הגלובליים הבאים על מנת שנוכל לעבוד עם הגדלים הנתונים הקבועים בקלות:

```
#define MEMORY_SPACE 512
char mem_arr[MEMORY_SPACE + 1][9];
int reg_arr[16];
static int memin_len;
static int pc = 0;
static int cycles = 0;
```

כאשר:

MEMORY_SPACE- הינו גודל הזיכרון (כמות השורות המקסימלי ב- memin,memout).

Mem_arr- הינו מערך דו מימדי שמטרתו להכיל את תוכן קובץ הקלט כמחרוזות באורך 8 ספרות הקסדצימליות.

Reg_arr- מערך מסוג int המכיל את תוכן הרגיסטרים במהלך ריצת התוכנית.

Memin_len- אורך קובץ הקלט.

Pc- משתנה מסוג int המאותחל ל-0, ומטרתו להכיל את הכתובת הרלוונטית במהלך הריצה.

Cycles- משתנה מסוג int המאותחל ל-0 ומטרתו לספור את כמות הפעולות המבוצעות במהלך הריצה.

<u>הגדרות מבנה נתונים הבאים:</u>

1) מבנה נתונים בשם struct המכיל את הפקודה המבוצעת (inst) ואת חלוקת שורת הפקודה בהתאם למבנה פקודה בקוד אסמבלי.

```
//defining command as a struct

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

לפני תחילת הריצה בפונקציית ה-main והפעלת הפונקציות בסימולטור, נכריז על הפונקציות בהן נשתמש במהלך הריצה:

```
// fuctions declaration
void command_fields(char* command_line, Command* com);
void Execute(Command* com, FILE* ptrace, FILE* pcycles, FILE* pmemout, FILE* pregout);
void Instructions(FILE* ptrace, FILE* pcycles, FILE* pmemout, FILE* pregout);
void Int_to_Hex_Arr(int num, char hexa[9]);
int Hex_char_to_int(char Hex);
int Hexa_Int_2s(char* hexnum);
void FillArray(FILE* memin);
void Trace(Command* com, FILE* ptrace);
void MemOut(FILE* pmemout);
void RegOut(FILE* pregout);
```

<u>אופן פעולת הסימולטור:</u>

ראשית, נבצע קליטה לקובץ הקלט memin.txt ממנו נקרא את הפקודות שהגיעו מהאסמבלר.
בנוסף, נבצע פתיחה גם לקבצי הפלט הנ"ל. לאחר מכן, נבדוק את תקינות הקבצים ובמידה וימצא קובץ לא תקין או חסר נקבל את ההודעה הבאה:

```
Error opening one or more of the files!
```

ונצא מהתוכנית עם קוד סיום 1 (המורה על תוכנית שלא בוצעה כראוי).
במידה והקבצים תקינים- תחילה נקרא לפונקציה FillArray המקבלת מצביע לקובץ הקלט ומעתיקה את תכולתו למשתנה mem_arr כשורות של מחרוזות ולאחר מכן נסגור את קובץ הקלט.
לאחר מכן נקרא לפונקציה Instructions שלמעשה תבצע את הפקודות אחת אחרי השניה ותממש בפועל את הפעולות הנדרשות מהסימולטור. נזכור בכל פעולה גם לכתוב את השורה הרלוונטית לקובץ הtrace.
לבסוף, נייצר את קבצי הפלט הנותרים בעזרת פונקציות MemOut,RegOut, נכתוב את כמות הפעולות לקובץ cycles ונסגור את כל הקבצים.

<u>הכרזות עבור כל הפונקציות בהן יעשה שימוש בתוכנית האסמבלר:</u>

```
// functions declaration
void command_fields(char* command_line, Command* com);
void Execute(Command* com, FILE* ptrace, FILE* pcycles, FILE* pmemout, FILE* pregout);
void Instructions(FILE* ptrace, FILE* pcycles, FILE* pmemout, FILE* pregout);
void Int_to_Hex_Arr(int num, char hexa[9]);
int Hex_char_to_int(char Hex);
int Hexa_Int_2s(char* hexnum);
void FillArray(FILE* memin);
void Trace(Command* com, FILE* ptrace);
void MemOut(FILE* pmemout);
void RegOut(FILE* pregout);
```

<u>הפונקציות בהן נשתמש:</u>

1) הפונקציה command_fields:

```
void command_fields(char* command_line, Command* com);
```

פונקציה זו מקבלת שורה המכילה פקודה כמחרוזת בבסיס הקסדצימלי באורך 8 ספרות.
הפונקציה מחלקת את הפקודה לפי השדות של מבנה הנתונים Command בהתאם למבנה פקודה בשפת אסמבלי.

2) הפונקציה Execute:

```
void Execute(Command* com, FILE* ptrace, FILE* pcycles, FILE* pmemout, FILE* pregout);
```

פונקציה זו הינה פונקציית הביצוע של הפקודות ע"י הסימולטור. הפונקציה מקבלת מצביעים לכל קבצי הפלט ובנוסף משתנה מסוג Command. תחילה נכתוב לקובץ הtrace את השורה הלוונטית מאחר וקובץ זה מעודכן לפני ביצוע כל פקודה. זאת נעשה ע"י פונקציית Trace עליה נסביר בהמשך. לאחר מכן, בהתאם לקוד בשדה opcode שב-Command נדע איזו פעולה עלינו לבצע בשורה זו. בכל פקודה שאנו מבצעים, נוסיף 1

לcycles. בנוסף, נעדכן את ה-pc בהתאם לפקודה המבוצעת.במידה ומתקבל פקודת halt, נייצר את קבצי הפלט, ונסגור את כל הקבצים ונסיים את הריצה.

- באופן כללי, התוכנית מסתיימת כאשר מבוצעות כל הפקודות או כאשר מבוצעת פקודת halt.

3) הפונקציה Instructions :

```
void Instructions(FILE* ptrace, FILE* pcycles, FILE* pmemout, FILE* pregout);
```

פונקציה זו הינה פונקציה העוברת על המערך mem_arr אליו כתבנו את כל שורות קובץ הקלט בפונקציית command_fields, ומעבירה כל פקודה לפונקציית Execute על מנת לבצע את הפקודה.

4) הפונקציות Int_to_Hex_Arr, Hex_char_to_int, Hexa_Int_2s :

```
void Int_to_Hex_Arr(int num, char hexa[9]);
int Hex_char_to_int(char Hex);
int Hexa_Int_2s(char* hexnum);
```

פונקציית Int_to_Hex_Arr הינה פונקציה המקבלת משתנה מסוג int וממירה אותו למחרוזת הקסדצימלית באורך 8. על הפונקציות Hex_char_to_int, Hexa_Int_2s הן פונקציות הזהות לפונקציות עליהן הוסבר באסמבלר הנ"ל.

5) הפונקציה FillArray :

```
void FillArray(FILE* memin);
```

פונקציה זו מקבלת מצביע לקובץ הקלט ומעתיקה את תוכנו למטריצה mem_arr המכילה מחרוזות בבסיס הקסדצימלי. במידה וקובץ הקלט אינו באורך 512 שורות, הפונקציה 'מרפדת' באפסים את שאר השורות.

6) הפונקציה Trace :

```
void Trace(Command* com, FILE* ptrace);
```

פונקציה זו נקראת בכל פקודה מחדש ע"י פונקציית Execute ומעדכנת את קובץ trace.txt עפ"י הפורמט :

PC INST R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 R15

כאשר כל ערך בשורה הנ"ל מייצג מחרוזת הקסדצימלית באורך 8 ספרות כאשר PC מייצג את הכתובת המבוצעת באותה פקודה, INST מייצג את הפקודה כמחרוזת הקסדצימלית באורך 8 ספרות, R0 מייצג את הקבוע 0 ולכן יהיה תמיד "00000000", R1 מייצג את ערך ה-immediate באותה פקודה. ושאר השדות מייצגים את ערכי הרגיסטרים טרם ביצוע הפקודה הנוכחית המבוצעת.

7) הפונקציות MemOut,RegOut :

```
void MemOut(FILE* pmemout);
void RegOut(FILE* pregout);
```

פונקציות אלה מייצרות את קבצי הפלט memout.txt, regout.txt בהתאמה ע"י כתיבת mem_arr, reg_arr לקבצים בהתאמה.

תוכניות הבדיקה

נציין שעבור כל תכנית בדיקה ביצענו תרגום מפונקציה הכתובה בשפת c . עבור כל תכנית נציג את הקוד בשפת c עבורו ביצענו את התרגום לשפת assembly.

1) תכנית הבדיקה summat.asm – תכנית שמטרתה לסכום שני מטריצות:

תכנית זו מבוססת על הקוד הבא :

```
void summat(int mat1[] ,int mat2[],int mat3[] ,int size){
    int start = 0;
    int end = size-1;
    int i=start;
    while (i <= end) {
        mat3[i]=mat1[i]+mat2[i];
        i++;
    }
}
```

ניתן לראות בתוכנית הבדיקה בשפת assembly בהערות ה comment# עבור כל שורה איזו פקודה מתאימה לה בקוד לעיל .בנוסף כתבנו תכנית המוודאת את תקינות הקוד הנ"ל בשפת C (ניתן לראות גם כי התרגום בmemout,בדוגמאות ההרצה זהה למה שמתקבל בתכנית זו) :

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

char* numberToHex(int number) {
    char* hexString = (char*)malloc(9 * sizeof(char)); // Allocate memory for 8 characters + null terminator
    // Format the number as a hexadecimal string
    snprintf(hexString, 9, "%08X", (unsigned int)number);

    return hexString;
}

void summat(int mat1[], int mat2[], int mat3[], int size) {
    int start = 0;
    int end = size - 1;
    while (start <= end) {
        mat3[start] = mat1[start] + mat2[start];
        start++;
    }
}

void print_mat(int mat[]) {
    for (int i = 0; i < 16; i++) {
        if (i != 0 && i % 4 == 0) {
            printf("\n");
        }
        printf(" %s ", numberToHex(mat[i]));
    }
}

int main() {
    int mat1[16] = { 5,31,46,12,7,25,17,200,-99,1,100,22,54,125,77,-66 };
    int mat2[16] = { 4,45,26,71,80,531,-13,201,3,99,41,63,-22,96,64,2 };
    int mat3[16];
    summat(mat1, mat2, mat3, 16);
    printf("mat1=\n");
    print_mat(mat1);
    printf("\nmat2=\n");
    print_mat(mat2);
    printf("\nmat1+mat2=\n");
    print_mat(mat3);
    return 0;
}
```

הפלט המתקבל הוא :

```
mat1=
 00000005  0000001F  0000002E  0000000C
 00000007  00000019  00000011  000000C8
 FFFFFF9D  00000001  00000064  00000016
 00000036  0000007D  0000004D  FFFFFFBE
mat2=
 00000004  0000002D  0000001A  00000047
 00000050  00000213  FFFFFFF3  000000C9
 00000003  00000063  00000029  0000003F
 FFFFFFEA  00000060  00000040  00000002
mat1+mat2=
 00000009  0000004C  00000048  00000053
 00000057  0000022C  00000004  00000191
 FFFFFFA0  00000064  0000008D  00000055
 00000020  000000DD  0000008D  FFFFFFC0
```

Output for summat.c code as we supposed to see in memout file

פרטים נוספים על תכנית summat.asm :

נחזיק ברגיסטר את הכתובת של האיבר הראשון במטריצה הראשונה וכן ברגיסטר נוסף את הכתובת של המקום האחרון.

אנו יודעים כי המטריצות שאותם אנו רוצים לחבר נמצאות אחת אחרי השנייה בזיכרון – כלומר מוקצים 16 מקומות למטריצה הראשונה ולאחריה 16 מקומות הבאים מכילים את המטריצה השנייה . כמו כן למטריצת התוצאה אשר נמצאת אחריהן מוקצים 16 מקומות נוספים בזיכרון . מכאן שהמרחק בין האיבר ה i במטריצה הראשונה לאיבר ה i במטריצה השנייה (כאשר 1<i<16) הוא בדיוק 16 לכל i . בנוסף, מההסבר לעיל המרחק בין האיבר ה i במטריצה הראשונה לאיבר ה i במטריצת התוצאה הוא בדיוק 32 .

לכן שנרצה לגשת לאיבר i במטריצה הראשונה לא נצטרך לעשות דבר ופשוט ניגש אליו ,כשנרצה לגשת לאיבר ה i במטריצה השנייה נוסיף ל i 16 ואז כשנרצה לאכסן את הסכום במטריצת התוצאה במקום ה- i שלה נוסיף לi 32 .

יש לחזור על הפעולה 16 פעמים על מנת לבצע את פעולת הסכימה של שני המטריצות מאינדקס ההתחלה של המטריצה הראשונה ועד סוף המטריצה הראשונה, כך שבסיום פעולת החיבור נעלה את אינדקס ההתחלה ב-1 , ולכן נדרשת לולאה כך שהתנאי לעצירה הוא שלא הגענו לסוף המטריצה הראשונה .

2) <u>תכנית הבדיקה bubble.asm</u> – תכנית שמטרתה למיין מערך של מספרים:

תכנית זו מבוססת על הקוד הבא:

```
void bubble(int array[] ,int size){
    int start = 0;
    int end = size-1;
    while (start <= end) {
        int i = start;
        while (i < end) {
            int temp;
            if (array[i] > array[i+1]) {    // if array[i] > array[i+1]
                temp = array[i];              // swap(array[i], array[i+1])
                array[i] = array[i+1];
                array[i+1] = temp;
            }
            i++;
        }
        end--;                            // decrement end
    }
}
```

ניתן לראות בתכנית הבדיקה בשפת assembly בהערות ה comment# עבור כל שורה איזו פקודה מתאימה לה בקוד לעיל . בנוסף כתבנו תכנית המוודאת את תקינות הקוד הנ"ל בשפת C (ניתן לראות גם כי התרגום בmemout, בדוגמאות ההרצה זהה למה שמתקבל בתכנית זו):

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

char* numberToHex(int number) {
    char* hexString = (char*)malloc(9 * sizeof(char)); // Allocate memory for 8 characters + null terminator

    // Format the number as a hexadecimal string
    snprintf(hexString, 9, "%08X", (unsigned int)number);

    return hexString;
}

void bubble(int array[], int size) {
    int start = 0;
    int end = size - 1;
    while (start <= end) {
        int i = start;
        while (i < end) {
            int temp;
            if (array[i] > array[i + 1]) {    // if array[i] > array[i+1]
                temp = array[i];              // swap(array[i], array[i+1])
                array[i] = array[i + 1];
                array[i + 1] = temp;
            }
            i++;
        }
        end--;                            // decrement end
    }
}


void print_array(int arr[]) {
    for (int i = 0; i < 16; i++) {
        printf("%s\n", numberToHex(arr[i]));
    }
}


int main() {
    int arr[16] = { 16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1 };
    printf("Before bubble sort:\n");
    print_array(arr);
    bubble(arr, 16);
    printf("\nAfter bubble sort:\n");
    print_array(arr);
    return 0;
}
```

הפלט המתקבל הוא :

```
Microsoft Visual Studio Debug Console
Before bubble sort:
00000010
0000000F
0000000E
0000000D
0000000C
0000000B
0000000A
00000009
00000008
00000007
00000006
00000005
00000004
00000003
00000002
00000001

After bubble sort:
00000001
00000002
00000003
00000004
00000005
00000006
00000007
00000008
00000009
0000000A
0000000B
0000000C
0000000D
0000000E
0000000F
00000010
```

Output for bubble.c code as we supposed to see in memout file

פרטים נוספים על תכנית bubble.asm :

נחזיק ברגיסטר את הכתובת של האיבר הראשון במערך וכן ברגיסטר נוסף את הכתובת של המקום האחרון .

פעולת המיון מתבצעת בשני לולאות. עבור הלולאה החיצונית עם התנאי start<=end – לולאה זו שולטת במספר המעברים הדרושים בפועל למיין את המערך . בכל איטרציה של הלולאה החיצונית, האלמנט הגדול ביותר בחלק אשר נותר לא ממוין מועבר למיקומו הנכון בקצהו. לאחר כל מעבר שכזה , טווח הלולאה הפנימית מצטמצם על ידי הקטנת הקצה. הסיבה לכך היא שהאלמנט הגדול ביותר בחלק הלא ממוין "מבעבע" עד הסוף לאחר כל מעבר, ואין לנו צורך לכלול אותו עוד באיטרציות הבאות.

הלולאה הפנימית מבצעת את ההשוואות וההחלפות בין אלמנטים סמוכים וכאשר הם בסדר לא נכון, הם מוחלפים. תהליך זה נמשך עד שלא מתקיים עוד מצב הלולאה הפנימית, דבר המצביע על כך שכל ההשוואות וההחלפות הנדרשות עבור המעבר הנוכחי בוצעו.

יחד, שתי הלולאות פועלות במקביל כדי לעבור שוב ושוב את המערך, להשוות אלמנטים סמוכים ולהחליף אותם במידת הצורך עד שהמערך כולו ממוין. הלולאה החיצונית שולטת במעברים, והלולאה הפנימית מבצעת את ההשוואות וההחלפות הנדרשות בתוך כל מעבר. כך שבסוף ריצת האלגוריתם, המערך ממוין בסדר עולה.

3) תכנית הבדיקה binom.asm – תכנית שמטרתה לחשב את המקדם הבינומי בהינתן n,k

תכנית זו מבוססת על קוד רקורסיבי שנכתב עבורנו בפרויקט :

```c
#include <stdio.h>
int binom(int n,int k){
if (k == 0 || n == k){
    return 1;
}
return binom(n-1, k-1) + binom(n-1, k);
}

int main()
{
    printf("%d",binom(7,4));

    return 0;
}
```

ניתן לראות בתכנית הבדיקה בשפת assembly בהערות ה #comment עבור כל שורה איזו פקודה מתאימה לה בקוד לעיל .

בנוסף כתבנו תכנית המוודאת את תקינות הקוד הנ"ל בשפת C (ניתן לראות גם כי התרגום בmemout,בדוגמאות ההרצה זהה למה שמתקבל בתכנית זו) :

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* numberToHex(int number) {
    char* hexString = (char*)malloc(9 * sizeof(char)); // Allocate memory for 8 characters + null terminator

    // Format the number as a hexadecimal string
    snprintf(hexString, 9, "%08X", (unsigned int)number);

    return hexString;
}


int binom(int n, int k) {
    if (k == 0 || n == k) {
        return 1;
    }
    return binom(n - 1, k - 1) + binom(n - 1, k);
}

int main()
{
    int n = 6;
    int k = 4;
    printf("n= %s\n", numberToHex(n));
    printf("n= %s\n", numberToHex(k));
    printf("n choose k = %s", numberToHex(binom(n, k)));

    return 0;
}
```

הפלט המתקבל הוא :

```
Microsoft Visual Studio Debug Console
n= 00000006
k= 00000004
n choose k = 0000000F
```

Output for binom.c code as we supposed to see in memout file

<u>פרטים נוספים על תכנית binom.asm :</u>

אתחול: הקוד מתחיל באתחול המשתנים n ו-k עם ערכים מכתובות הזיכרון 256 ו-257, בהתאמה כפי שמצוין בפרויקט. מכיוון שמדובר בקוד רקורסיבי נצטרך לעבוד עם מחסנית על מנת לאחסן את הקריאות הרקורסיביות לפונקציה. בתחילת הריצה כל הרגיסטרים מאותחלים לאפס ולכן יש לאתחל את $sp לערך אחר.נבחר את מיקום המחסנית באופן כזה כך שלא תהיה דריסה של הקוד או הדאטא.

לאחר האתחול נעבור על ידי פקודת jal לפרוצדורה הבאה:

קריאה לLable בשם: binom – בLable זה נשמור במחסנית את כל הערכים שאותם נרצה לשחזר עבור הקריאות הרקורסיביות כדוגמת n,k והערך שאליו נרצה לחזור ב($ra) בסיום כל קריאה רקורסיבית. לאחר מכאן נבדוק את תנאי העצירה ובמידה ומתקיים אחד מהם נעבור לLable בשם Base אחרת נעבור לLable בשם L1.

הקוד תחת ה Lable בשם L1 :

זוהי הפונקציה המרכזית, בהנחה ותנאי העצירה לא מתקיימים נבצע קריאות רקורסיביות כפי שמבוטא בקוד בשפת c . ואת התוצאה מאחסנים ב $v0 .

הקוד תחת ה Lable בשם Base :

בהנחה ואחד מתנאי העצירה מתקיים נעבור לLable בשם Base אשר בו נחזיר $v0=1 . בסיום נעבור לLable בשם L2 שמטרתו לבצע "ניקוי" של המחסנית על ידי הוספת 4 ולחזור לשורה המתאימה לאחר הקפיצה באמצעות $ra .

לבסוף, את הערך השמור ברגיסטר $v0 נרצה לשמור בזיכרון על ידי פקודת sw בכתובת 258 כפי שהתבקשנו, ולאחר מכאן התכנית תסתיים עם פקודת halt .

דוגמאות הרצה:

1) תכנית הבדיקה summat.asm:

ניתן לראות כאן בmemout את המטריצה הראשונה והשנייה מאוחסנות בזיכרון במקומות המתאימים (השורה המסומנת היא האיבר הראשון של המטריצה השנייה) עבור קלט שכולל כל מקרה אפשרי (מספרים שליליים וחיובים בכתיב דצימלי והקסדצימלי, אותיות גדולות או קטנות, בכתובת או בערך):

```
# Initialize mat1
.word 256 5
.word 257 31
.word 258 46
.word 259 12
.word 260 7
.word 261 25
.word 262 17
.word 263 200
.word 264 -99
.word 265 0x000000C
.word 266 100
.word 267 22
.word 268 54
.word 269 125
.word 270 77
.word 271 -66
```

```
# Initialize mat2
.word 272 4
.word 273 45
.word 274 26
.word 275 71
.word 276 80
.word 277 531
.word 278 -13
.word 279 201
.word 0x118 3
.word 281 99
.word 282 0xffffffff
.word 283 63
.word 284 -22
.word 285 96
.word 286 64
.word 287 2
```

כעת נציג את תוצאות החיבור ואת מיקומי המטריצות כדי לוודא שהם בהתאם לקוד:

```
00000005
0000001F
0000002E
0000000C
00000007
00000019
00000011
000000C8
FFFFFF9D
0000000C
00000064
00000016
00000036
0000007D
0000004D
FFFFFFBE
Ln 257, Col 9
```

```
00000004
0000002D
0000001A
00000047
00000050
00000213
FFFFFFF3
000000C9
00000003
00000063
FFFFFFFF
0000003F
FFFFFFEA
00000060
00000040
00000002
Ln 273, Col 9
```

```
00000009
0000004C
00000048
00000053
00000057
0000022C
00000004
00000191
FFFFFFA0
0000006F
00000063
00000055
00000020
000000DD
0000008D
FFFFFFC0
Ln 289, Col 9
```

ואכן מתקיים שכל איבר הוא חיבור של שני האיברים המתאימים במטריצות – למשל חיבור שני האיברים הראשונים 5+4=9, או שני האיברים השניים 45+31=76 – שזה שווה לC4 בהקס.

מספור השורות גדול ב1 מהערך שביקשו כי הספירה בקובץ מתחילה ב1 ולא ב0.

2) <u>תכנית הבדיקה bubble.asm</u>:

נריץ על מערך ממויין מהגדול לקטן עבור קלט שכולל כל מקרה אפשרי (מספרים שליליים וחיובים בכתיב דצימלי והקסדצימלי, אותיות גדולות או קטנות, בכתובת או בערך) ונראה שאכן מתקבל את אותם ערכים ממויינים מקטן לגדול:

```
# Initialize array
.word 256 16
.word 257 15
.word 258 14
.word 259 13
.word 0x104 17
.word 261 11
.word 262 0xFffffffff
.word 263 9
.word 264 0x0000000C
.word 265 7
.word 266 -2
.word 267 5
.word 268 4
.word 269 3
.word 270 2
.word 271 1
```

```
FFFFFFFE
FFFFFFFF
00000001
00000002
00000003
00000004
00000005
00000007
00000009
0000000B
0000000C
0000000D
0000000E
0000000F
00000010
00000011
Ln 257, Col 9
```

3) <u>תכנית הבדיקה binom.asm</u>:

נריץ תחילה עם n=5, k=3, n choose k = 10 ואכן מתקבל בmemout תוצאה נכונה בשורות המתאימות:

```
# Initialize n,k
.word 256 5
.word 257 3
```

```
00000005
00000003
0000000A
Ln 257, Col 9
```

ניתן לראות כי n שזה 5 בדוגמא זו ממוקם בשורה המתאימה, לאחר מכן שורה אחרי ממוקם k שזה 3 בדוגמא זו, ותוצאת הn choose k מתחתיהן.

דוגמא נוספת: n=6, k=4, n choose k= 15

```
# Initialize n,k
.word 256 6
.word 257 4
```

```
00000006
00000004
0000000F
Ln 257, Col 9
```

ניתן לראות כי n שזה 6 בדוגמא זו ממוקם בשורה המתאימה, לאחר מכן שורה אחרי ממוקם k שזה 4 בדוגמא זו, ותוצאת הn choose k מתחתיהן.