אסמבלר :

- בכתיבת האסמבלר בנינו שני "structures" שיעזרו לנו בבניית הקוד :

1. Labels:
   - Labels_name.
   - Address: is the label's " Label's_name" Pc number.
2. Words:
   - Place: Data Memory Place.
   - Value.

- בכתיבת האסמבלר חילקנו את העבודה לכמה פונקציות קצרות שכל אחת עושה את החלק שלה, הפונקציות שהשתמשנו בהן היו :-

1. add_word:
   - input: array of words(structure) "words[]",integer Data memory place,integer WordCounter,char* Value.
   - output: the new number of words "integer WordCounter
2. fill_words:
   - input: File* fp_out ,integer Word_counter,array of words(structure) "words[]".
   - output: the function copies the values in Word's array to fp_out file and doesn't return anything.
3. checkReg:
   - input: char* Reg.
   - output: the function decodes the Register number and returns the register number as string as in the register's table we have .
4. checkOpcode:
   - input: char* opcode,array of char (string) "opcodenum[]".
   - output: the function decodes which opcode the string opcode represents and doesn't return a thing.
5. islabel:
   - input: char* CheckLabel, integer Label_counter, array of Labels "Labbels[]".
   - output: If the string Check label is a label the function returns 1 otherwise 0.
6. GetLabelsAdress:
   - Input: char* labelName, integer label_counter,array of labels "Labels[]"
   - Output: if we have LabelName in the array of labels "labels[]" return his address otherwise "-1".
7. GetwordssValue:
   - Input: array of Words(structure) "words[]",integer place,integer Word_counter.
   - Output: the function return's the value that is in the place "place" in the array "words" if we have it otherwise return's -1.
8. read_labels:
   - input: File fp_in, array of Label(structure) "Labels".
   - output: the function reads the assembly code and fills the labels that in it in the array "Labels" and returns the array's length.
9. opcodeprint:
   - input: FILE* fp_out,5 char* : opcode, imm, rd, rs, rt, integer Label_Counter, array of Label(structure) Labels[].
   - output: the function print the instruction we have to the output file fp_out and doesn't return a thing.

10. read_asm:
    - input: FILE* fp_in, FILE* fp_out,integer word_counter, integer Label_Counter,array of labels(structure) "Labels[]".
    - output: the number of words the function "saw" .
11. maxWordPlace:
    - input: array of words(structure)"words,integer Word_Count
    - output: the farther place in the array with value bigger than zero.

- **<u>תהליך עבודת האסמבלר:</u>**

1. הקוד שלנו עושה קריאה ראשונה על קוד האסמבלי שנמצא בקובץ הקלט וממלא מערך "Labels" לפי שם ה"Label" והמיקום שלו.
2. הקוד מייצר מערך של "Words" שבסוף התהליך הוא יכיל את תמונת זיכרון הנתונים שצריך להעתיק אותו לקובץ "Dmemin"
3. הקוד עובר בקריאה שניה על הקוד ומנתח את כל המצבים האפשריים ("Label", ".word", "opcode") :
   - אם מדובר בפקודת ".word" אזי הוא מעדכן את הערך המתאים שלה במערך "Words".
   - אם מדובר ב"Label" אזי הוא עובר לפקודה הבאה.
   - אם מדובר ב"opcode" אזי הוא מפענח את השורה ומחלק אותה לפי ההנחיות ל5 סיביות היקסדיצמליות כך ששתי הספרות השמאליות ביותר יהוו הערך המתאים לאופקוד שנקרא ושלוש הספרות הבאות יהיו מספרי הרגיסטרים שאובחנו על ידי הפונקציה "CheckReg" ואם בבדיקת הפונקציה "ImmNeed" אובחן שהפקודה משתמשת ב"Immediate" אזי ממירים את הערך שלו ל5 ספרות היקסדיצמליות ומדפיסים הכל לפי הסדר הנדרש לתוך הקובץ "Imemin" שמהווה את זיכרון ההוראות .
4. בסוף, הקוד מעתיק את התוכן של המערך "Words" לזיכרון הנתונים וסוגר את כל הקבצים.

- **<u>עמידה בדרישות:</u>**

1. הקוד שלנו מתעלם מכל ה"white spaces" .
2. הנחנו שאורך השורה המקסימאלי בקבצי הקלט הוא 500.
3. הנחנו שאורך ה"Label" הוא 50.
4. כתבנו הערות בקובץ הקוד ליד כל פונקציה וליד כל שורה שחשבנו שהיא קצת לא ברורה וצריך הערה .
5. הקוד עובד עם פקודות ".word" שלא נמצאות רק בסוף הקוד אלא כל מקום אפשרי.
6. הקוד מוציא הודעת שגיאה מתאימה עבור כל מקרה שצריך כגון : בעיה בפתיחת קובץ ,מספר ה"command arguments" פחות מ4 ...

סימולטור:

- **בכתיבת האסמבלר חילקנו את העבודה לכמה פונקציות קצרות שכל אחת עושה את החלק שלה, הפונקציות שהשתמשנו בהן היו :**

1. chechReg:
   - input: char Reg.
   - output: the function decodes the register number from the register's table we have and returns this value.
2. checkIOReg:
   - input: integer IORegin, char array regWRname[].
   - output: the function decodes which input/output register we are dealing with and filles its number in the array regWRname[] and doesn't return a thing.
3. checkopcode:
   - input: char array opcode[]
   - output: the function decodes the opcode number we are dealing with from the opcode's table we have and returns this value.
4. opcodeImm:
   - input: FILE* fp_Ledo, FILE* fp_hwRegtrace,7 integers : opcode,rd ,rs ,rt, imm,pcc,clk_count, array of integers clk[],array of integers Dmem[],array of integers R[],array of integers IOReg[],array of integers screen[288][352],array of char Disk[2562][9].
   - output: the function handles the instruction we have print's if we have to to the File we should print to and returns the pc value.
5. opcodeNoImm:
   - input: FILE* fp_Ledo, FILE* fp_hwRegtrace,7 integers : opcode,rd ,rs ,rt, imm,pcc,clk_count, array of integers clk[],array of integers Dmem[],array of integers R[],array of integers IOReg[],array of integers screen[288][352],array of char Disk[2562][9].
   - output: the function handles the instruction we have print's if we have to to the File we should print to and returns the pc value.
6. isjump:
   - Input: integer opcode.
   - Output: the function checks if we are dealing with branch instruction if that is the situation it returns 1 otherwise returns zero.
7. isInOut:
   - Input: char* opcode.
   - Output: the function checks if we are dealing with Input or Output instruction if that is the situation it returns 1 otherwise returns zero.

8. immneed:

- input: char rs, char rd, char rt.
- output: the function checks if we are dealing with an instruction that includes immediate number if that is the situation it returns 1 otherwise returns zero.

9. copyImem:

- input: FILE* fp_Imem, char Imem[][MAX_ROW].
- output: the function copies the instructions from the input file to Imem array and returns the amount of instructions we have.

10. copyDMem:

- input: FILE* fp_Dmem ,array of integers Dmem[].
- output: the function copies the Data memory from the input file to Dmem array and returns the amount of Data we have.

11. CopyCLK:

- input: FILE* fp_irq2,array of integers CLK.
- output: the function copies the clock cycles that irq2 goes up to "1" from the input file to the array CLK and return the amount of clock cycles we are dealing with.

12. copyDisk:

- input: FILE*fp_Diskin, array of integers Disk[].
- output: the function copies the Disk data from the input file to the array Disk[] and doesn't return anything.

13. irq2Setup:

- input: integer irq2,integer clk_count, array of integers clk[].
- output: function that checks if we have irq2 in the array clk and returns 1 if we have it otherwise returns 0.

14. MaxData:

- input: array of integers Dmem.
- output: the function return the maximum place that the Data in it is bigger than zero.

15. fill_data:

- input: FILE* fp_out , array of integers Dmem[],integer Dmem_Counter.
- output: the function updates the data memory(output file) to the values we have in the Data array.

16. fill_Disk:

- input: FILE* fp_out, array of integers Disk[2560][9].
- output: the function updates the Disk data(output file) to the values we have in the Disk array.

17. ReadOrWrite:

- Input: char* opcode, char* typeWR.
- Output: the function decodes if we are reading or writing and updates the string typeWR with the name of the read or write instruction that opcode points to(from the opcode table we have).

18. Print_Regout:

- Input: FILE* fp_regout ,array of integers R[].
- Output: the function prints the status of registers 2-16 to the output file "fp_regout".

19. Print_Regout4trace:

- input: File* fp_regout, array of integers R[], integer pc, char* inst.
- output: the function prints the status of register 2-16 before applying the instruction and the instruction we have right now and the number of instruction we have.

20. HWRegtrac:

- input: FILE* fp_hwr, integer cycle ,integer data, char* name, char* type.
- output: the function print to the output file the cycle number we have now and the name of IORegister and the data stored in the IO registers .

21. PrintscreenStatus:

- input: FILE* fp_screen, array of integers screen[288][352].
- output: the function updates the screen status from the screen array to the output file in hexadecimal base.

22. PrintScreenStatusBinary:

- input: FILE* fp_screen, array of integers screen[288][352].
- output: the function updates the screen status from the screen array to the output file in binary base.

23. WriteLeds:

- input: FILE* fp_ledo, array of integers leds[8],integer cycle.
- output: the function that prints the status of the leds and the clock cycle we are in.

24. CyclesTimePc:

- input: FILE* fp_cycletime ,integer pc,integer time.
- output: the function prints the amount of instructions the simulator "did" and the amount of clock cycles it takes to "did it" to the output file.

- **<u>תהליך עבודת הסימולטור:</u>**

הסימולטור מגדיר כמה מערכים : "Imamin" ומעתיק אליו את תוכן זיכרון ההוראות , "Dmamin " ומעתיק אליו את תוכן זיכרון הנתונים,"Diskin " ומעתיק אליו את תוכן הדיסק הקשיח,"Clk" ומעתיק אליו את תוכן מחזורי השעון שעבורן קו הפסיקה החיצוני "Irq2" עלה ,ואז עובר על הוראה הוראה זיכרון ההוראות ו"עושה "אותה ובכל פעם שהוא "שרואה" שהתרחשה פסיקה אזי שומר את מספר ההוראה שהוא אמור לעשות ב "irqreturn" וקופץ להוראה לנמצאת ב"irqhandler" ואחרי שיגיע לפקודת"reti" חוזר וממשיך את העבודה שלו כרגיל (בכל פעם שהוא אמור להדפיס לקובץ כלשהו תוך כדי ריצתו הוא מדפיס כמובן)וכשיסיים את כל ההוראות ידפיס לכל הקבצים שהוא אמור להדפיס לתוכן בסיום הריצה שלו ואז יסגור את כל הקבצים .

- **<u>עמידה בדרישות:</u>**

1. הנחנו שאורך השורה המקסימאלי בקבצי הקלט הוא 500.
2. הנחנו שאורך ה"Label" הוא 50.
3. הנחנו שאורך זיכרון הנתונים המקסימאלי בקבצי הקלט הוא 4096.
4. הנחנו שאורך זיכרון ההוראות המקסימאלי בקבצי הקלט הוא 1024.
5. כתבנו הערות בקובץ הקוד ליד כל פונקציה וליד כל שורה שחשבנו שהיא קצת לא ברורה וצריך הערה .
6. הקוד מוציא הודעת שגיאה מתאימה עבור כל מקרה שצריך כגון : בעיה בפתיחת קובץ ,מספר ה"command arguments" פחות מ13 ...
7. ברגע שמתרחשת פסיקה ישירות הקוד שלנו קופץ ו"עושה" אותה וחוזר להמשיך (מהמקום שיצא ממנו) את ההוראות שנשארו לו .
8. הקוד שלנו ברגע שעובד עם פסיקה של דיסק קשיח לוקח 1024 מחזור שעון כמו שהתבקש.
9. עקב בעיה (בגודל זיכרון המחשב שלנו) לא היה ביכולתנו להגדיר את גודל הדיסק הקשיח ל16384 (המחשב קרס) ולכן הגדרנו אותו להיות בגודל של 20 סקטור ולא 128 כלומר 2560.
10. בדקנו על ידי התוכנה שהוצעה לנו שאכן מצויר מעגל על המסך כנדרש.

Test Functions:

1. Bubble :

- הסבר :הקוד שלנו מאתחל את הערכים של המערך לכתובות המתאימות לפי הדרישות שהיא מ"1024" עד"1039" על ידי 16 פקודת "Word.", ואז ממין אותם לפי שיטת "Bubble Sort" הידועה.

2. Binom:

- הסבר : הקוד שלנו מאתחל את הערכים של "n" ו- "k" על ידי שתי פקודות "Word." ואז מחשב את בינום ניוטון המבוקש.

3. Disktest:

- הסבר : הקוד שלנו עובר על שלושת הסוקטורים הראשונים ומעדכן את המיקום המתאים בסיקטור 4 לפעולת "XOR" על כל הערכים במקומות המתאימים בשלושת הסקטורים הראשונים מ0 עד 128.

4. Mulmat:

- הסבר :הקוד שלנו מאתחל את הערכים של כל מטריצה למקומות המתאימים מ"0x100" עד"0x10F" ומ"0x110" עד"0x11F"בזיכרון הנתונים על ידי 32 פקודת "Word." ואז מכפיל את שתי המטריצות לפי כפל מטריצות רגיל ושומר את התוצאות המטריצה שמתחילה מ"0x120" ועד "0x12F".

5. Leds:

- הסבר : הקוד שלנו משתמש בפסיקת הקו החיצוני "irq2" כדי לייצור השהייה של שניה בין הדלקת כל נורה וכדי לסיים בדיוק אחרי הדלקת כל הנורות כלומר אחרי 63 מחזור שעון, ובכל פעם שהפסיקה מתרחשת הוא לוקח את המצב הנוכחי של הלדים ואש על ידי "Shift Right" ו" Bitwise Or" עם "0x70000000" מדליקים את הנורה הבאה .

6. Circle:

- הסבר :הקוד שלנו מאתחל את רדיוס העיגול לכתובת "0x100" על ידי פקודת "word." ואז עובר על כל הרדיוסים מ0 עד R ועל כל הXים מ0 עד 352 וכל הYים מ0 עד288 ובכל פעם שמתקיימת משוואת המעגל הידועה "$(x-175)^2+(y-143)^2=R^2$" (לפי המרכז המבוקש)מעדכן את ה "monitorx=x" ו"monitory=y" " ו"monitordata=255" ו"monitorcmd=1" ומעלה את הקו "irq0enable=1" ו"irq0status=1" " לl .

עמידה בדרישות הגשה:

בספרייה כוללת שהגשנו יש:

❖ **שתי ספריות שונות :**

- עבור האסמבלר :יש את "asm" שכוללת בתוכה את הקוד ב"visual studio" וקובץ ה"solution" שניתן לבנות אותו על ידי לחיצה על "build solution" בנוסף לספריית ה"build" שכוללת את קובץ ה"executable" הבנוי.
- עבור הסימולטור :יש את "sim" שכוללת בתוכה את הקוד ב"visual studio" וקובץ ה"solution" שניתן לבנות אותו על ידי לחיצה על "build solution" בנוסף לספריית ה"build" שכוללת את קובץ ה"executable" הבנוי.

❖ **ועוד 5 ספריות עבור פונקציות הבדיקה :**

1. "mulmat" : שיש בתוכו 16 קבצים הכוללים עותק של קבצי "asm.exe","sim.exe" ואת קבצי הקלט והפלט של ריצת תוכנית הבדיקה דרך האסמבלר והסימולטור.
2. "bubble" : שיש בתוכו 16 קבצים הכוללים עותק של קבצי "asm.exe","sim.exe" ואת קבצי הקלט והפלט של ריצת תוכנית הבדיקה דרך האסמבלר והסימולטור.
3. "binom" : שיש בתוכו 16 קבצים הכוללים עותק של קבצי "asm.exe","sim.exe" ואת קבצי הקלט והפלט של ריצת תוכנית הבדיקה דרך האסמבלר והסימולטור.
4. "leds" : שיש בתוכו 16 קבצים הכוללים עותק של קבצי "asm.exe","sim.exe" ואת קבצי הקלט והפלט של ריצת תוכנית הבדיקה דרך האסמבלר והסימולטור.
5. "clock" : שיש בתוכו 16 קבצים הכוללים עותק של קבצי "asm.exe","sim.exe" ואת קבצי הקלט והפלט של ריצת תוכנית הבדיקה דרך האסמבלר והסימולטור.
6. "disktest" : שיש בתוכו 16 קבצים הכוללים עותק של קבצי "asm.exe","sim.exe" ואת קבצי הקלט והפלט של ריצת תוכנית הבדיקה דרך האסמבלר והסימולטור.

❖ והשארנו גם את ספריית הבדיקה שקיבלנו אותה כתובה "fib" עם 16 קבצים הכוללים עותק של קבצי "asm.exe","sim.exe" ואת קבצי הקלט והפלט של ריצת תוכנית הבדיקה דרך האסמבלר והסימולטור.

❖ וקובץ הדוקומנטציה הזה כמובן.

הערות:

❖ במחשב שכתבנו את הקוד של האסמבלר והסימולטור הייתה בעיה בהורדת "visual studio" ,כך שלא מצליח לכתוב לקבצי "txt" , הקוד עובד כמו שצריך בכל מחשב אחר לכן יש סיכויי שבספריית האסמבלר והסימולטור כשמריצים אותו ללא "cmd" ידפיס לקבצי "asm".