דוקומנטציה פרויקט מבנה המחשב סמסטר ב תשפ"ד

מגישים: שחר שובל (318724994), עדי מסלטי (209057785) ושני בר יוסף (207487497)

בפרויקט זה פיתחנו סימולטור ואסמבלר עבור מעבד RISC בשם SIMP. הפרויקט נועד להעמיק את ההבנה שלנו בנושאים של תכנון מעבדים, כתיבת תוכניות בשפת אסמבלי ותרגול שפת C.
הסימולטור מסמלץ את פעולתו של מעבד ומכיל גישה למספר התקני קלט/פלט. במסגרת הפרויקט נדרשנו לכתוב תוכניות בשפת אסמבלי, לתכנן וליישם אסמבלר שמתרגם את הקוד לשפת מכונה, ולבנות סימולטור שמריץ את הקוד על גבי סימולציית המעבד שיצרנו.

The Assembler

האסמבלר- נכתב בשפת C ותפקידו לתרגם תוכניות אסמבלי עבור המעבד הרצוי לשפת מכונה.
האסמבלר יקבל כקובץ קלט קובץ באסמבלי (sort, binom, triangke, discktest) ויחזיר כקובץ פלט את תמונת הזיכרון בקובץ memin.txt.

האסמבלר מחולק ל3 חלקים- הגדרות ומבנים, פונקציות עזר ופונקציות עיקריות:

הגדרות ומבנים-

כולל את כל ההגדרות המקדימות והמבנים שבהם משתמש האסמבלר כדי לפרש ולתרגם את קוד האסמבלי לשפת מכונה. בחלק זה מוגדרים הקבועים, המשתנים, המערכים והאופרטורים שישמשו במהלך פעולת האסמבלר.

חלק זה מכיל הגדרות של אורך השורות המקסימלי, אורך התוויות, ואורך הזיכרון, מכיל מערכי מחרוזות המגדירים את האופקודים והרגיסטרים. זאת ועוד, נעזרנו ב- enum שממפה את האופקודים לשמותיהם ונעזרנו גם במבני נתונים למטרת ניהול זיכרון, מערך דו-ממדי לזיכרון הפלט final_memומערכים לשמות וערכי הלייבלים בקוד האסמבלי - label_names ו- label_values.

פונקציות עזר-

find_and_check_register- הפונקציה מזהה את שם הרגיסטר ובודקת שאכן תקין ולאחר מכן בודקת האם הרגיסטר מיידי. הפונקציה מקבלת כפרמטרים-

- char reg[]- שם הרגיסטר
- int dest_or_arg- רגיסטר ארגומנט או יעד
- int opcode- האופקוד של ההוראה

והפונקציה מחזירה (נעזרת גם בפונקציית עזר check_if_imm) 1 אם הרגיסטר הוא מיידי ו 0 אם לא.

_find_reg- פונקציה זו מקבלת כפרמטר שם של רגיסטר (char reg[]) ומחזירה את המספר הסידורי שלו (0 עד 15), אם השם אינו מוכר התוכנית מחזירה 16.

check_if_imm- פונקציה זו בודקת אם הרגיסטר מיידי.

הפונקציה מקבלת כפרמטרים-

- int reg_num- מספר סידורי של הרגיסטר
- int dest_or_arg- רגיסטר ארגומנט או יעד
- int opcode- האופקוד של ההוראה

הפונקציה מחזירה 1 אם הרגיסטר הוא מיידי ו 0 אם לא.

פונקציות עיקריות- מבצעות את המשימות המרכזיות של האסמבלר: קריאת קובץ האסמבלי, ניתוח המלל, תרגום הקוד לשפת מכונה וכתיבת הפלט לקובץ.

read_asm- פונקציה לקריאת קובץ אסמבלי, זיהוי מידע ועדכון מבנים. מחזירה את אורך השורות בקוד.

עובדת בצורה הבאה- פותחת קובץ האסמבלי לקריאה, קריאת כל שורה בקובץ ובדיקת סוג השורה (תווית, נתונים או הוראה). כעת על פי הבדיקה, אם מדובר בהוראה- שמירת ההוראה במערך הקוד ובדיקת שימוש האם מיידי; אם מדובר בתווית/לייבל- שמירת שם וכתובת הזיכרון.לסיום סוגרת את הקובץ ומחזירה את אורך השורות בקוד שקראה.

פרמטרים-

- int* label_len-מצביע למספר התוויות .
- int* last_addr-מצביע לכתובת האחרונה בזיכרון שבה יש נתונים.
- int label_values[]- מערך של הערכים ,כתובות, של התוויות.
- char code[MEM_LEN][MAX_LABEL_LEN + 50]- מערך דו-ממדי שמכיל את שורות הקוד האסמבלי הנקי.
- char final_mem[MEM_LEN[6]- מערך דו-ממדי שמייצג את הזיכרון הסופי של המעבד.
- char label_names[MEM_LEN][MAX_LABEL_LEN + 1]- מערך דו-ממדי שמכיל את שמות התוויות.
- char asm_file[]- שם קובץ האסמבלי לקריאה.

parse_labels- מפרשת את התוויות ומעדכנת את כתובות הזיכרון שלהן, מבצעת תרגום של כל הוראה והוראה לשפת מכונה ומעדכנת את הזיכרון.

פונקציה זו עוברת על כל שורת קוד שנקראה בפונקציה read_asm , מזהה את האופקוד והאופרנדים בכל שורה. לאחר מכן מתרגמת את האופקוד והאופרנדים לערך בינארי. במידה ויש שימוש ב -אופרנד מיידי הפונקציה מעדכנת את הערך המתאים בזיכרון. לבסוף, הפונקציה מעדכנת את הכתובת האחרונה בזיכרון שבה יש נתונים מתוך מטרה לשמור על מעקב מדויק אחרי מצב הזיכרון.

פרמטרים-

- int input_len- אורך הקוד בקובץ האסמבלי.
- int label_len- מספר התוויות שנמצאו.
- int last_addr- הכתובת האחרונה בזיכרון עם נתונים.
- int label_values[]- מערך של הערכים ,כתובות, של התוויות.
- char label_names[MEM_LEN][MAX_LABEL_LEN + 1]- מערך דו-ממדי שמכיל את שמות התוויות.
- char code[MEM_LEN][MAX_LABEL_LEN + 50]- מערך דו-ממדי שמכיל את שורות הקוד האסמבלי הנקי.
- char final_mem[MEM_LEN[6]- מערך דו-ממדי שמייצג את הזיכרון הסופי של המעבד.

מוחזרת הכתובת האחרונה בזיכרון שבה יש נתונים.

dump_mem- תפקידה הוא לכתוב את תוכן הזיכרון לקובץ הפלט memin.txt.

הפונקציה פותחת את קובץ הפלט לכתיבה, עוברת על כל הכתובות בזיכרון עד לכתובת האחרונה שבה יש נתונים, וכותבת את התוכן של כל תא זיכרון לקובץ הפלט. לבסוף, הפונקציה סוגרת את קובץ הפלט.

הפלט של הפונקציה הוא קובץ טקסט המכיל את תוכן הזיכרון לאחר עיבוד של המעבד. כל שורה בקובץ מייצגת תא זיכרון ומכילה מחרוזת באורך 5 תווים בבינארי שמתארת את הערך המאוחסן באותו תא זיכרון.

פרמטרים-

- int last_addr- כתובת האחרונה בזיכרון שבה יש נתונים (מציין את גבול הכתובות שצריך לכתוב לקובץ הפלט).
- char final_mem[MEM_LEN[6]- מערך דו-ממדי שמייצג את הזיכרון הסופי של המעבד.
- char mem_file[]- שם קובץ הפלט שאליו ייכתב תוכן הזיכרון.

main- פונקציה זו כשמה היא הפונקציה הראשית, היא מהווה ככניסה לתוכנית. היא אחראית על קריאה לפונקציות העיקריות האחרות במטרה לבצע את תהליך האסמבלר מההתחלה ועד הסוף, מהקריאה של קובץ האסמבלי ועד לכתיבת קובץ הפלט.

הפונקציה קוראת לפונקציה read_asm כדי לקרוא ולפרש את קובץ האסמבלי. לאחר מכן, היא קוראת לפונקציה parse_labels כדי לפרש את הלייבלים ולתרגם את הקוד לשפת מכונה. לבסוף, הפונקציה קוראת לפונקציה dump_mem כדי לכתוב את תוכן הזיכרון שעבר עיבוד לקובץ הפלט.

פרמטרים-

- int argc- מספר הארגומנטים של שורת הפקודה.
- char* argv[]- מערך המחרוזות של הארגומנטים של שורת הפקודה.

<u>כמובן</u> הפלט הסופי של התוכנית יהיה קובץ הזיכרון (memin.txt) שבו כל שורה מייצגת תא זיכרון המכיל קוד בינארי שנוצר מהקוד האסמבלי.

The Simulator

הsimulator אחראי לסמלץ את מעבד הSIMP וכמו כן מספר התקני קלט/ פלט: נורות, תצוגת ה -7 segment, מוניטור מונוכרומטי ברזולוציה של 256X256 ודיסק קשיח. נפרט על פונקציות העזר שכתבנו בחלק של הsimulator.

void CheckArg(int argc);

הפונקציה מקבלת את מספר הארגומנטים שמשתמשים בהם כאשר קוראים לתוכנית ומוודאת שזהו המספר אליו אנו מצפים. במידה ולא, הפונקציה עוצרת את התוכנית. הפונקציה לא מחזירה כלום.

ארגומנטים:

int argc- argc פרמטר מסוג integer, מייצג את מספר הארגומנטים שמשתמשים בהם כאשר קוראים לתוכנית.

void LoadFiles(char* Filenames[], FILE* files[]);

הפונקציה מקבלת מערך של handles לקבצים, ומערך של שמות הקבצים הנ"ל ופותחת את הקבצים אחד אחרי השני. הפונקציה לא מחזירה כלום.

ארגומנטים:

char* Filenames[]- מערך של שמות הקבצים. המערך הנ"ל הוא למעשה הארגומנט של התוכנית. לפיכך, השימוש בארגומנט בתוך הפונקציה מתחיל מהאיבר השני המערך שכן האיבר הראשון הוא השם של התוכנית.

FILE* files[]- מערך של handles לקבצים. שלושת הקבצים הראשונים הינם קבצי קריאה ואילו שאר הקבצים הינם קבצי כתיבה.

int ReadIRQ2Cycles(int irq2_cycles, FILE* irq2in);**

הפונקציה מקבלת מצביעים למערך irq2_cycles שמחזיק את המחזורים בהם irq2 מעלה פסיקה (interrupt). בנוסף הפונקציה מקבלת את מצביע לקובץ irq2in שמכיל את הקלטים של irq2 ושומרת אותם במערך irq2_cycles. הפונקציה מחזירה את אורך המערך irq2_cycles.

ארגומנטים:

int irq2_cycles** - מצביעים למערך irq2_cycles שמחזיק את המחזורים בהם irq2 מעלה פסיקה (interrupt).

FILE* irq2in – מצביע מסוג FILE לקובץ irq2in, המכיל את הקלטים של irq2.

void Readdisk(int disk[], FILE* diskin);

הפונקציה מקבלת מערך של הדיסק ומצביע לקובץ הדיסק ומעתיקה את הערך המספרי של תוכן קובץ הדיסק למערך הדיסק. הפונקציה לא מחזירה כלום.

ארגומנטים:

int disk[] – מערך שאבריו משתנים מסוג int, שם המערך הינו disk, והוא מייצג את הדיסק הקשיח.

FILE* diskin – מצביע מסוג FILE לקובץ diskin.

void Readmem(int mem[], FILE* memin);

הפונקציה מקבלת מערך הזיכרון ומצביע לקובץ הmemin ומעתיקה את הערך המספרי של תוכן קובץ הmemin למערך הזיכרון. הפונקציה לא מחזירה כלום.

<u>ארגומנטים:</u>

int mem[] – מערך בשם mem שאבריו מסוג int.

FILE* memin – מצביע מסוג FILE לקובץ memin.

void CheckIRQ(int* PC, int irq, int IO_regs[], int* is_irq_handler);

הפונקציה מקבלת מצביע PC האינדיקטור irq, מערך הIO reg ואינדיקטור הirq handler. בתחילת כל מחזור שעון הפונקציה בודקת האם קפץ לנו irp ועל סמך זאת הפונקציה מחליטה האם הPC הבא יצטרך לקפוץ לirqhandler. אם אכן הPC קופץ, הפונקציה מעדכנת את הIO reg הרלוונטי עם כתובת חזרה מעודכנת. הפונקציה לא מחזירה כלום.

<u>ארגומנטים:</u>

int* PC – מצביע למשתנה מסוג int בשם PC. מכיל את הכתובת להוראה הבאה.

int irq – משתנה מסוג int בשם irq. משמש לפסיקות.

int IO_regs[]– מערך בשם IO_regs שאבריו מסוג int.

int* is_irq_handler – מצביע למשתנה מסוג int בשם is_irq_handler.

void IO_update(int IO_regs[], int* cycle_for_disk_end, int* irq2_cycles, int irq2_len);

הפונקציה מקבלת את מערך הIO_regs , מצביע למשתנה cycle_for_disk_end עם המחזור והדיסק איתם האינטראקציה אמורה להסתיים ועם עם הפסיקות של irq2. בנוסף הפונקציה מקבלת מצביע לirq2_cycles שהוא מערך המחזורים ומשתנה irq2_len שמכיל את אורכו של מערך המחזורים. הפונקציה מחליטה האם לשנות את הIO_regs הרלוונטי במערך הנ"ל לפי מספר מחזורי השעון והמחזור בו אמורה להתרחש הפסיקה. הפונקציה לא מחזירה כלום.

<u>ארגומנטים:</u>

int IO_regs[]– מערך בשם IO_regs שאבריו מסוג int.

int* cycle_for_disk_end – מצביע למשתנה מסוג int בשם cycle_for_disk_end.

int* irq2_cycles – מצביע למשתנה מסוג int בשם irq2_cycles.

int irq2_len –משתנה מסוג int בשם irq2_len.

void LoadInstructionData(int mem[], int* opcode, int* rd, int* rs, int* rt, int* imm, int* PC, FILE* trace);

הפונקציה מקבלת את מערך הmem , מצביעים לassembly line arguments עם כל הרגיסטרים שהפונקציה תצטרך לקרוא מהזיכרון, מצביע למשתנה PC ומצביע לקובץ trace. הפונקציה משמשת לקריאת את המידע מהזיכרון. הפונקציה לא מחזירה כלום.

ארגומנטים:

int mem[]– מערך בשם mem שאבריו מסוג int. מערך הזיכרון.

int* opcode – מצביע למשתנה מסוג int בשם opcode. מחזיק מידע בקשר לסוג ההוראה אותה רוצים לבצע.

int* rd – מצביע למשתנה מסוג int בשם rd. רגיסטר היעד.

int* rs – מצביע למשתנה מסוג int בשם rs. רגיסטר עליו מתבצעת ההוראה.

int* rt – מצביע למשתנה מסוג int בשם rt. רגיסטר עליו מתבצעת ההוראה.

int* imm – מצביע למשתנה מסוג int בשם imm.

int* PC – מצביע למשתנה מסוג int בשם PC. מכיל את הכתובת להוראה הבאה.

FILE* trace – מצביע מסוג FILE לקובץ trace.

void ExecuteCommand(int Opcode, int rd, int rs, int rt, int imm, int* PC, int* is_irq_handler, int* is_halt, int* cycle_for_disk_end, int RegArray[],int IO_regs[], int mem[], int monitor[256][256], int disk[], FILE* trace, FILE* hwregtrace, FILE* leds_file_name, FILE* display);

הפונקציה אחראית לביצוע פקודה. הפונקציה מקבלת את הארגומנטים של קטע הקוד, הPC, את הirq handler, את הdisk end ואינדיקטורי העצירה, את מערך הreg ומערך הIO reg, הזיכרון, המוניטור והדיסק ובנוסף מקבלת כמה קבצים. בתחילת אחראית ביצוע הוראה (פקודה) ובהתאם לפקודה מעדכנת את הקבצים, המערכים והאינדיקטורים הרלוונטיים. הפונקציה לא מחזירה כלום.

ארגומנטים:

int opcode – משתנה מסוג int בשם opcode. מחזיק מידע בקשר לסוג ההוראה אותה רוצים לבצע.

int rd – משתנה מסוג int בשם rd. רגיסטר היעד.

int rs – משתנה מסוג int בשם rs. רגיסטר עליו מתבצעת ההוראה.

int rt – משתנה מסוג int בשם rt. רגיסטר עליו מתבצעת ההוראה.

int imm – מצביע למשתנה מסוג int בשם imm.

int* PC – מצביע למשתנה מסוג int בשם PC. מכיל את הכתובת להוראה הבאה.

int* is_irq_handler – מצביע למשתנה מסוג int בשם is_irq_handler.

int* is_halt – מצביע למשתנה מסוג int בשם is_halt.

int* cycle_for_disk_end – מצביע למשתנה מסוג int בשם cycle_for_disk_end.

int RegArray[] – מערך שאבריו מסוג int בשם RegArray.

int IO_regs[] – מערך שאבריו מסוג int בשם IO_regs.

int mem[] – מערך שאבריו מסוג int בשם mem. מערך הזיכרון.

int monitor[256][256] – מערך דו ממדי, מטריצה ריבועית מסדר 256 על 256 שאיבריה הם משתנים מסוג int. שם המערך monitor, והוא מייצג את המוניטור המונוכרומטי ברזולוציה של 256X256.

int disk[] – מערך שאבריו משתנים מסוג int, שם המערך הינו disk, והוא מייצג את הדיסק הקשיח.

FILE* trace – מצביע מסוג FILE לקובץ trace.

FILE* hwregtrace – מצביע מסוג FILE לקובץ hwregtrace.

FILE* leds_file_name – מצביע מסוג FILE לקובץ leds_file_name.

FILE* display – מצביע מסוג FILE לקובץ display.

void inout_inst(int in_or_out, int rd, int rs, int rt, int* cycle_for_disk_end, int RegArray[], int IO_regs[], int monitor[256][256], int disk[], int mem[], FILE* hwregtrace, FILE* leds_file_name, FILE* display);

הפונקציה מקבלת את אינדיקטור הin או out, שורת הקוד באסמבלי שכוללת את הopcode והרגיסטרים עליו מתבצעת הפעולה ורגיסטר היעד. בנוסף הפונקציה מקבלת את המחזור עבור מבציע סיום הדיסק, את המערך RegArray, את מערך הIO_regs , מערך המוניטור, מערכי הדיסק והזיכרון, ואת המצביעים לקבצי הleds , hwregtrace ,ו display . הפונקציה אחראית לעדכן את מערך הIO_regs לפי שורת הקוד באסמבלי שהיא מקבלת. בנוסף, הפונקציה מעדכנת את האינדיקטורים, המערכים והקבצים שקיבלה בהתאם לשורת הקוד הנ"ל. הפונקציה לא מחזירה כלום.

<u>ארגומנטים:</u>

int in_or_out – משתנה מסוג int בשם in_or_out . מחזיק מידע בקשר לסוג האינדיקטור שהפונקציה מקבלת: in או out.

int rd – משתנה מסוג int בשם rd. רגיסטר היעד.

int rs – משתנה מסוג int בשם rs. רגיסטר עליו מתבצעת ההוראה.

int rt – משתנה מסוג int בשם rt. רגיסטר עליו מתבצעת ההוראה.

int* cycle_for_disk_end – מצביע למשתנה מסוג int בשם cycle_for_disk_end.

int RegArray[]– מערך שאבריו מסוג int בשם RegArray.

int IO_regs[] – מערך שאבריו מסוג int בשם IO_regs.

int monitor[256][256] – מערך דו ממדי, מטריצה ריבועית מסדר 256 על 256 שאיבריה הם משתנים מסוג int. שם המערך monitor, והוא מייצג את המוניטור המונוכרומטי ברזולוציה של 256X256.

int disk[] – מערך שאבריו משתנים מסוג int, שם המערך הינו disk, והוא מייצג את הדיסק הקשיח.

int mem[] – מערך שאבריו מסוג int בשם mem. מערך הזיכרון.

FILE* hwregtrace – מצביע מסוג FILE לקובץ hwregtrace.

FILE* leds_file_name – מצביע מסוג FILE לקובץ leds_file_name.

FILE* display – מצביע מסוג FILE לקובץ display.

void finish_program(int mem[], int disk[], int Reg_Array[], int IO_regs[], int monitor[256][256], FILE* memout, FILE* regout, FILE* cycles, FILE* diskout, FILE* monitor_file, FILE* monitoryuv);

הפונקציה מקבלת את כל המערכים של התוכנית, את המוניטור וכמה קבצים. הפונקציה אחראית לשלוח את תוכן המערכים לתוך הקבצים ומפסיקה בערך האחרון ששונה מאפס. הפונקציה לא מחזירה כלום.

<u>ארגומנטים:</u>

int mem[] – מערך שאבריו מסוג int בשם mem. מערך הזיכרון.

int disk[] – מערך שאבריו משתנים מסוג int, שם המערך הינו disk, והוא מייצג את הדיסק הקשיח.

int RegArray[]– מערך שאבריו מסוג int בשם RegArray.

int IO_regs[] – מערך שאבריו מסוג int בשם IO_regs.

int monitor[256][256] – מערך דו ממדי, מטריצה ריבועית מסדר 256 על 256 שאיבריה הם משתנים מסוג int. שם המערך monitor, והוא מייצג את המוניטור המונוכרומטי ברזולוציה של 256X256.

FILE* memout – מצביע מסוג FILE לקובץ memout. מכיל את תוכן הזיכרון הראשי בסיום הריצה.

FILE* regout – מצביע מסוג FILE לקובץ regout. מכיל את התוכן הרגיסטרים בסיום הריצה

FILE* cycles – מצביע מסוג FILE לקובץ cycles. מכיל את מספר מחזורי השעון שהתוכנית רצה.

FILE* diskout– מצביע מסוג FILE לקובץ diskout.

FILE* monitor_file – מצביע מסוג FILE לקובץ monitor_file.

FILE* monitoryuv– מצביע מסוג FILE לקובץ monitoryuv.

void print_to_last_val(int mem[], int len, FILE* file);

הפונקציה מקבלת את מערך הזיכרון ואורכו יחד עם מצביע לקובץ file אליו הפונקציה משליכה את המערך. הפונקציה אחראית להדפיס את המערך לקובץ file, עד הערך האחרון שהוא איננו אפס. הפונקציה לא מחזירה כלום.

<u>ארגומנטים:</u>

int mem[] – מערך שאבריו מסוג int בשם mem. מערך הזיכרון.

int len –משתנה מסוג int בשם len. אורכו של מערך הזיכרון.

FILE* file– מצביע מסוג FILE לקובץ file.

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

...

}

בדומה לקובץ הassembler , תפקידה של פונקציית הmain הוא הפונקציה הראשית בה מתבצעות כלל הקריאות לפונקציות העזר שפורטו לעיל, בסיום הריצה נוצרים קבצי הפלט הנדרשים. פונקצית הmain מחזירה 0 אם ריצתה התבצעה בצורה תקינה.

<u>ארגומנטים:</u>

int argc – argc פרמטר מסוג integer, מייצג את מספר הארגומנטים שמשתמשים בהם כאשר קוראים לתוכנית.

char* argv[] – מערך של משתנים מסוג char בשם argv. מייצג את מערך המחרוזות של הארגומנטים של שורת הפקודה.

תוכניות בדיקה

תוכנית ,sort.asm אשר מבצעת מיון bubble sort של 16 מספרים בסדר עולה .המספרים נתונים בכתובות 100 x 0עד x10f 0,ואלו גם כתובות המערך הממוין בסיום.
בעצם בקוד זה הגדרנו t_0 להיות האידיקטור אם ביצענו החלפה בכל איטרציה.
הפונקציה תיעצר לאחר שהגענו ל-15 איברים.

תוכנית ,binom.asm המחשבת את מקדם הבינום של ניוטון באופן רקורסיבי לפי האלגוריתם הבא. בתחילת הריצה n נתון בכתובת 0x100, k בכתובת 0x101 והתוצאה תיכתב לכתובת 0x102. ניתן להניח כי n מספיק קטן כך שאין overflow. תוכנית זו ממומשת באופן רקורסיבי על ידי שימוש בstack pointer.

תוכנית ,triangle.asm שמציירת על המסך משולש ישר זווית מלא בצבע לבן (כל הפיקסלים בהיקף ובתוך שטח המשולש לבנים). כאשר A הינו הקודקוד השמאלי העליון, זווית B היא זווית ישרה, ו C - הינו הקודקוד הימני התחתון. צלע AB הינה אנכית, וצלע BC הינה הצלע האופקית .כתובות קודקודי המשולש יחסית לתחילת ה frame buffer -נתונות בכתובות הבאות:
0x100(A),0x101(b),0x102(c) . תחילה הבחנו כי מערכת הצירים הפוכה.
הקוד מתמקד בחישוב השיפועים של שני קווים עיקריים AB ,ו BC-ומבצע השוואת בין השיפועים ומציאת הקווים המתאימים ביותר לציור על המסך. בעצם מתבצע מעבר על כל הערכים האפשריים שהשיפוע יכול לקבל ונבדוק מי הכי קרוב בערך מוחלט.
נקודה חשובה היא שהמעבד שלנו לא תומך בפעולות חילוק לכן הינה דוגמה אחת מבין הנוסחאות לחישוב השיפוע: $m(x_2 - x_1) = y_2 - y_1$.
בעצם חיפשנו את m_1 m_2 שמקיימים $m_2(y - y_1) = m_1(x_1 - x_2)$ וביצענו לולאה בתוך לולאה כך שמחפשים את m_1 ואת m_2 האופציונליים, עוברים על הערכים מ1-256 ומוצאים את המינימום.
לאחר מכן נוכל לבדוק אם הנקודה נמצאת בטווח.

תוכנית ,disktest.asm שמבצעת סיכום של תוכן סקטורים 0 עד 7 בדיסק הקשיח וכותבת את תוצאת הסיכום לסקטור מספר 8, כלומר כל מילה בסקטור 8 תהיה סכום 8 המילים המתאימות מסקטורים 0 עד 7 .
בעצם נקרא ל 8 סקטורים הראשונים מהזיכרון ,נסכום אותם ואז נוציא לסקטור ה-9 בדיסק.