**איפיון פרוקיט מעבדה**

פרויקט זה, נבנה ע"י איתי בלדב ת.ז: וכן ע"י שותפו לפרויקט מקס פרימן ת.ז:

בקובץ נפרט בקצרה (ברמת הסבר) את כל הקבצים והפעולות אותם בנינו ואת מבנה הפרויקט:

מבנה הפרויקט בקצרה:

חילקנו את הפרויקט למספר חלקים, בחלק הראשון אנו מבצעים את הקדם אסמבלר(שבו יש לנו גם טבלת המאקרוים), שבו אנו בעצם נפטרים בכל הגדרות המאקרו, ופורשים אותם לקוד אמיתי.

לאחר מכן אנו עוברים על הקובץ הפרוש משלב הקדם האסמבלר, ומתחילים לבנות את טבלת הסמלים. אם נמצאה שגיאה אנו נמשיך לעבור על כל הקובץ בשביל למצוא שגיאות נוספות, ולבסוף נעצור. חשוב לציין כי בחרנו לא לקודד כלל בחלק זה אלא רק לאתר שגיאות ולבנות את טבלת הסמלים.

לבסוף נבצע מעבר נוסף על הקובץ הפרוש, רק שהפעם יש בידינו טבלת סמלים בנויה, ואת ההנחה שהקובץ תקין (פרט לשגיאות שניתן למצוא אך ורק במעבר השני).

בחלק זה אנו בעצם מקודדים, ואנו עושים זאת על ידי קידוד משפטי הוראה בקובץ נפרד (cmd), ועל ידי קידוד משפטי הנחיה בקובץ נפרד( data) , לבסוף אנו משלבים את קבצים אלו לקובץ ob אחד, אשר בהתחלה שלו יש את תוכן הקובץ cmd, ובסופו את תוכן הקובץ data.

בו זמנית אנו כותבים לקבצים ext,ent , כמבוקש בחוברת הקורס.

תיעוד קבצי הפרויקט:

Assembler.c

הקובץ העיקרי בפרויקט, לו כמובן אין קובץ.h והוא האחד שמפעיל את האלגוריתם הכתוב בחוברת.

שגיאות/הערות/פלט למסך/ קבצי תוצאה סופית ייצאו כולם ממנו בלבד.

יכלול בנוסף את הפעולות הבאות:

* symbolTable createSymbolTable(char\* fileName, flags\* status)

מקבלת פוינטר לקובץ ומבצעת עליו מעבר ראשון של האסמבלי, בזאת שבודקת את תקינות הקובץ, ובונה את טבלת הסמלים למעבר השני. בנוסף היא מקבלת מצביע למבנה שבו יש לנו דגלים(כמו האם נמצאה שגיאה, האם נמצאה הנחיית entry ועוד)

* int encodeAssembly(char\* fileName, symbolTable table, flags\* status)

מקבלת פוינטר לקובץ ומבצעת עליו מעבר שני של האסמבלי, היא מניחה שהקובץ תקין (פרט לשגיאות כמו תווית שלא הוגדרה, שכן היא זאת שאחראית על מציאתה), ומקודדת את הקובץ לקבצי הפלט ob, ent, ext .

**GeneralFunctions.c**

קובץ שבו נמצאות כל הפונקציות עזר שאנחנו משתמשים בקבצים השונים.

בתחילת הקובץ מוגדרים defines של הדפסות בצבעים.

הפונקציות שיש בה הן:

void getFileWithExtension(char\* fileName, char\* extension, char\* fileWithExtension)

פונקציה זו לוקחת שם קובץ וסיומת שלו, ומרכזת אותו ב-fileWithExtension.

הפרמטרים שהיא מקבלת הם:

file\_name - שם הקובץ

extention - הסיומת של הקובץ

fileWithExtension - המחרוזת שאליה יש לכתוב

* FILE\* openFile(char\* file\_name, char\* file\_extention, char\* mode)

פונקציה זו פותחת קובץ עם הסיומת המבוקשת לפי הנחיות הפרויקט

הפרמטרים שהיא מקבלת הם:

file\_name - שם הקובץ

file\_extention - הסיומת של הקובץ

mode - המצב איתו יש לפתוח את הקובץ

מחזירה מצביע של הקובץ שנפתח

* void deleteFile(char\* file\_name, char\* file\_extension)

פונקציה זו לוקחת שם קובץ וסיומת שלו, ומוחקת את הקובץ.

הפרמטרים שהיא מקבל הם:

file\_name - שם הקובץ

file\_extension - הסיומת של הקובץ

לא מחזירה דבר

* void removeSpacesAndTabs(char line[MAX\_LINE\_LENGTH])

פונקציה זו מוחקת את הרווחים מתחילה וסוף של מחרוזת.

הפרמטר שהיא מקבלת הוא:

line - המחרוזת שיש לשנות(עם הגודל המקסימלי עבור שורה כמו שביקשו במטלה)

לא מחזירה דבר

* void throwError(char\* errorMsg, int numberOfLine)

פונקציה זו זורקת שגיאה ומדפיסה אותה על המסך.

הפרמטרים שהיא מקבלת הם:

errorMsg - הודעת השגיאה שיש להדפיס

numberOfLine - מספר השורה בה התרחשה השגיאה

לא מחזירה דבר

* void throwWarning(char\* warningMsg, int numberOfLine)

פונקציה זו זורקת אזהרה ומדפיסה אותה על המסך.

הפרמטרים שהיא מקבלת הם:

warningMsg - הודעת האזהרה שיש להדפיס

numberOfLine - מספר השורה בה התרחשה השגיאה

לא מחזירה דבר

* void reservedWord (char\* word)

פונקצייה זו בודקת אם המילה הנתונה הינה מילה שמורה באסמבלי.

הפרמטרים שהיא מקבלת הם:

word – המילה לבדיקה

* int foundEmptySentence (char line[MAX\_LINE\_LENGTH])

פונקציה זו בודקת אם שורה מכילה רק תויים ריקים (רווחים/טאבים)

הפרמטר שהיא מקבלת הוא:

line - השורה לבדיקה(עם הגודל המקסימלי עבור שורה כמו שביקשו במטלה)

מחזירה 1 אם מצאה תו שהוא לא ריק, אחרת מחזירה 0

* int firstCharIsDot(char \*line)

פונקציה זו בודקת אם התו הראשון בשורה הוא נקודה

הפרמטרים שהיא מקבלת הם הפרמטר שהיא מקבלת הוא:

line - השווה לבדיקה

אם היא מוצאת נקודה בתחילת המחרוזת מחזירה 1 אחרת מחזירה 0

* void cutColonFromLabel(char \*labelName)

פונקציה זו מוחקת את הנקודתיים בסוף התווית

הפרמטר שהיא מקבלת הוא:

labelName – שם התווית

לא מחזירה דבר

* int isNumber(char \*number)

פונקציה זו בודקת אם מחרוזת מייצגת מספר חוקי

הפרמטר שהיא מקבלת הוא:

number - המחרוזת שיש לבדוק

אם המחרוזת מייצגת מספר חוקי אז תחזיר 1 אחרת תחזיר 0

* int convertToNumber(char\* numberString, int\* number)

פונקציה זו ממירה מחרוזת נתונה ל-int

הפרמטרים שהיא מקבלת הם:

numberString - המחרוזת שיש להמיר

number - מצביע למקום אליו נרצה להמיר

אם זה הצליח להמיר את המחרוזת תחזיר 0 אחרת תחזיר 1

**macroTable.c:**

בקובץ זה נמצאות כל הפונקציות לטיפול הרשימה מקושרת חד כיוונית שבה יש מידע על כל מאקרו. מבנה כל חוליה ברשימה הוא כזה:

char \*name

int startIndex;

int length;

struct Mnode\* next;

כאשר name מייצג את שם המאקרו, startIndex מייצג את מיקומו של התחלת המאקרו בקובץ, length מייצג את אורך המאקרו (בתויים), וnext מצביע לאיבר הבא ברשימה המקושרת.

הפונקציות שלה והפעולות שלהם:

* macroNode createMacroNode(char\* name)

פונקציה זו משמשת להקצאת זיכרון וליצירת חוליית מאקרו חדש

הפרמטר שהיא מקבלת הוא:

name - שם המאקרו

מחזירה מצביע למאקרו הריק החדש

* void InsertMacroNode(struct MNode\*\* head, struct MNode\* newNode)

פונקציה זו משמשת להכנסת מאקרו חדש לרשימת המאקרו

הפרמטרים שהיא מקבלת הם:

head – מצביע לראש הנוכחי של הרשימה המקושרת, אנו צריכים מצביע שכן אנו רוצים לשנות אותו.

newNode- המאקרו החדש שמוסף לרשימה

לא מחזירה דבר

* struct MNode\* SearchNode(struct MNode\* head, char \*name)

פונקציה זו משמשת לחיפוש מאקרו ספציפי ברשימה מקושרת

הפרמטרים שהיא מקבלת הם:

head - הראש הנוכחי של הרשימה המקושרת

name - השם לחיפוש

מחזירה את הצומת שנמצא, אחרת מחזירה NULL אם לא ניתן למצוא אותו

* void freeMacroList(struct MNode\* node)

פונקציה זו משמשת לשחרור רשימת המאקרו מזיכרון הheap

הפרמטר שהיא מקבלת הוא:

node - הצומת הנוכחי ברשימה המקושרת

לא מחזירה דבר

* int isValidMacroName (char\* name)

פונקציה זו משמשת לבדיקת תקינות שם של מאקרו, לפי התנאים שיקיר כתב בפורום.

הפרמטר שהיא מקבלת הוא:

name – השם לבדיקה

מחזירה 1 אם השם תקין, אחרת 0

* getName (struct MNode\* node), getStartIndex(struct MNode\* node), getLength(struct MNode\* node)

פונקציות אלו אחראיות על קבלת שדות המאקרו.

הפרמטר שהן מקבלות הוא:

node – מצביע לstruct של מאקרו, שלו אנו רוצים לגשת לשדות.

מחזירות את השדה המבוקש.

* setLength (struct MNode\* node, int length), setStartIndex(struct MNode\* node, int startIndex)

פונקציות אלו אחראיות על שינוי שדות המאקרו.

הפרמטר שהן מקבלות הוא:

node – מצביע לstruct של מאקרו, שלו אנו רוצים לגשת לשדות.

Length/startIndex – זה יהיה המשתנה החדש שאותו נשנה, בהתאם לפונקצייה.

לא מחזירות כלום

**PreAssembler.c:**

קובץ הקדם אסמבלר שלנו שבו מתבצעת פרישת המאקרוים מקבצי המקור לתוך קבצי

am

הפונקציות שלה:

* void WriteMacroToOutputFile(macroNode macro, FILE\* inputFile, FILE\* outputFile)

פונקציה זו משמשת לכתיבת התוכן של מאקרו לקובץ הפלט

הפרמטרים שלה הם:

macro - המאקרו שייכתב

- inputFile קובץ קלט המקור

- outputFile קובץ הפלט am לכתוב אליו

לא מחזירה דבר

* int addToMacroList(macroNode\* head, char\* line, FILE\* inputFile)

פונקציה זו משמשת כדי ליצור מאקרו חדש ברשימה המקושרת, על ידי חילוץ שמו, ותוכנו מקובץ הקלט.

. - head ראש הרשימה המקושרת של טבלת המאקרו

line – השורה הנוכחית בקובץ הקלט

- inputFile קובץ מקור הקלט

תחזיר 1 אם נמצאה שגיאה, אחרת 0

* int preAssemble(char\* file\_name)

פונקציה זו היא הראשית בשלב הקדם אסמבלר, והיא אחראית על יצירת קובץ am עם מקרואים פרושים.

הפרמטר שלה הוא:

- file\_name שם קובץ הקלט

תחזיר 0 אם הכל הלך כשורה, ואם נמצאה שגיה תחזיר 1.

**RegistgerTable.c:**

הוא קובץ המכיל מידע, ופונקציות עזר על טיפול ברגיסטרים

בתחילתו הגדרנו מערך גדול, שבו מוגדרים כל הרג'יסטרים , ומספרם.

בנוסף הקובץ מכיל פונקציות לטיפול ברגיסטרים:

* int isRegisterName(char \*name)

הפונקציה בודקת אם השם שהתקבל הוא שם של רגיסטר חוקי או לא

הפרמטר שלה הוא:

name - השם לחיפוש

מחזירה 1 אם היא מצאה את השם בטבלה אחרת מחזירה 0

* int getRegisterNum(char \*name)

הפונקציה בודקת אם השם שהתקבל הוא שם של רגיסטר חוקי או לא ומחזירה את המספר שלו

הפרמטר שלה הוא:

name - השם לחיפוש

מחזירה את מספר הרגיסטר אם היא מצאה את השם בטבלה אחרת מחזירה 0

**AddressingMode.c:**

הוא קובץ שמטפל בשמות ההנחיות וסוגי המיעונים השונים.

בתחילתו הוא מכיל מערך שבו ישנם את כל ההנחיות התקינות.

הפונקציות שיש בו הן:

* int isInstructionName(char \*name)

פונקציה זו בודקת אם מחרוזת נתונה היא שם של הנחיה חוקית בטבלת ההנחיות.

הפרמטר שלה הוא:

name - המחרוזת שיש לבדוק

אם מצאה את ההנחיה בטבלת ההנחיות היא תחזיר 1 אחרת תחזיר 0

* int getFirstDelimIndex(char \*str, char delim)

פונקציה זו מחזירה את האינדקס של ההופעה הראשונה של המפריד אותו אנו מחפשים

הפרמטרים שלה הם:

name - המחרוזת שיש לחפש עליה

delim - המפריד לחיפוש

מחזירה את האינדקס של המופע הראשון של המפריד הנתון או 0 אם הוא לא נמצא

* enum addressingMode getAddressingMode(char \*operand, int numberOfLine)

פונקציה זו מחזירה את סוג המיעון של האופרנד הנתון

הפרמטרים שלהם הם:

operand - האופרנד כדי לקבל את שיטת המיעון שלו

numberOfLine - השורה הנוכחית בקובץ הקלט

מחזירה את סוג המיעון של האופרנד הנתון אם לא מצאה כזה תחזיר error(1-)

בנוסף בקובץ h של הקובץ הזה יש לנו מבנה מסוג enum שמאחסן את סוגי המיעונים לפי המספרים שהם מקבלים כך הוא נראה:  
  
typedef enum addressingMode

{

מיעין מידי\\ immediateAddress = 0,

מיעון ישיר\\ directAddress = 1,

מיעון גישה לרשומה \\ addressAccess = 2,

מיעון רגיסטר ישיר \\ directRegisterAddress = 3,

שגיאה \\ error = -1

} addressingMode;

**SymbolTable.c:**

בקובץ זה אנו מטפלים בכל מה שקשור לסמלים בפרוייקט.

אנו מימשנו את טבלת הסמלים כרשימה מקושרת חד כיוונית של סמלים, וכל סמל מכיל את שמו, את את הכתובת שלו, ואת סוגו.

typedef struct symbol\* symbolTable;

/\*\* Next symbol in table \*/

symbolTable next;

/\*\* Address of the symbol \*/

int address;

/\*\* name (symbol name) is a string (aka char\*) \*/

char \*name;

/\*\* Symbol type \*/

symbolType type;

};

הפונקציות שלה והפעולות שלהם:

* int isLabel(char \*name)

פונקציה זו בודקת אם הסמל שהתקבל מסתיים בנקודתיים

הפרמטר שלה הוא

name – שמו של הסמל

מחזירה 1 אם נמצאו נקודתיים בתו האחרון של המחרוזת אחרת 0

* int validLabelName(char \*name)

פונקציה זו, בודקת את תנאי הסמל כפי שהוגדרו במטלה : אם התו הראשון הוא אלפא וכל השאר הם אלפאנומריים, והמחרוזת אינה מילה שמורת ושאורכה אינו ארוך יותר מ30 תווים

הפרמטר של הוא:

name - המחרוזת שיש לבדוק

מחזירה 1 אם כל התנאיים הנ"ל התקיימו אחרת מחזירה 0

* void setType(symbolTable symbol,symbolType type)

פונקציה זו, מכניסה לטבלת הסמלים את טיפוס הסמל

הפרמטרים שלה הם:

symbol – הסמל לו אנו מעדכנים את הטיפוס

type – הטיפוס שאותו הסמל יקבל

לא מחזירה דבר

* void setAddress(symbolTable symbol, int address)

פונקציה זו, מכניסה לטבלת הסמלים את כתובת הסמל

הפרמטרים שלה הם:

symbol – הסמל לו אנו מעדכנים את הכתובת

address – הכתובת שאותו הסמל יקבל

לא מחזירה דבר

* symbolTable findInTable(symbolTable symbol, char \*name)

פונקציה זו, מחפשת בטבלת הסמלים סמל השם שהיא קיבלה

הפרמטרים שלה הם:

symbol – הסמל הנוכחי שאותו אנו בודקים (בהתחלה הוא יהיה הhead, וזה ישתנה מכיוון שהפונקצייה פועלת רקורסיבית)

name – המחרוזת שאני משווים לסמל שאנו מחפשים

אם מצאנו את הסמל טבלת הסמלים היא תחזיר אותו אחרת תחזיר NULL

* symbolTable createSymbol(char\* name,int address)

פונקציה זו, יוצרת סמל בטלת הסמלים

הפרמטרים שלה הם:

name – מחרוזת שהיא שמו של הסמל

address – הכתובת שאותו הסמל יקבל

מחזירה את התא החדש שנוצר בטבלת הסמלים עם כל השדות שלו.

* void shiftHead(symbolTable\* head)

פונקציה זו, מזיזה את הראש של הרשימה המקושרת אחד קדימה, כלומר מוחקת את הhead.

הפרמטר שלה הוא:

head– התא שצריך להזיז

לא מחזירה דבר

* void InsertSymbolNode(symbolTable\* head, char \*label, int address)

פונקציה זו משמשת להכנסת סמל חדש לטבלת הסמלים

הפרמטרים שהיא מקבלת הם:

head - הראש הנוכחי של הרשימה המקושרת

Label - מחרוזת שמו של הסמל החדש

address – כתובתו של הסמל החדש

לא מחזירה דבר

* int getAddress(symbolTable symbol)

פונקציה זו, מקבלת מטבלת הסמלים את כתובת הסמל המצורף

הפרמטר שלה הוא:

symbol – הסמל שאנו ניגשים לו לכתובת

מחזירה את כתובת הסמל

* symbolType getType(symbolTable symbol)

פונקציה זו, מקבלת מטבלת הסמלים את טיפוס הסמל המצורף

הפרמטר שלה הוא:

symbol – הסמל שאנו ניגשים לו לטיפוס

מחזירה את סוג הסמל

* char\* getSymbolName(symbolTable symbol)

פונקציה זו, מקבלת מטבלת הסמלים את שם הסמל

הפרמטר שלה הוא:

symbol – הסמל שאנו ניגשים לו לשם

מחזירה את שם הסמל

* void updateDataSymbols(symbolTable table, int IC)

פונקציה זו, מעדכנת את מונה ההוראות עם מונה הנתונים של הסמלים בהתאם לטיפוס שלהם כמו שהתבקשנו במטלה.

הפרמטרים שלה הם:

table – טבלת הסמלים שאת סמליה אנו רוצים לעדכן

IC – מונה ההוראות

לא מחזירה דבר.

* void freeNode(symbolTable node)

פונקציה זו, משחררת תא זיכרון מטבלת הסמלים(הרשימה המקושרת שלנו)

הפרמטר שלה הוא:

node – תא הזיכרון ברשימה המקושרת שנרצה לשחרר

לא מחזירה דבר.

* void freeTable(symbolTable table)

פונקציה זו, משחררת מהזיכרון את טבלת הסמלים שיצרנו

הפרמטר שלה הוא:

table – טבלת הסמלים שנרצה לשחרר מהזיכרון

לא מחזירה דבר.

**AssemblySentence.c:**

קובץ זה מטפל בקובץ am אחרי פרישת המקרואים   
בקובץ זה נעבור שורה שורה על הקובץ וננתח מילה מילה בו ונפעל בהתאם.

בנוסף, בתחילת הקובץ header הגדרנו מבנה נתונים בשם flags, המכיל flags של מידע עבור המעבר הראשון. הוא מכיל שדה error, שאומר אם נמצאה שגיאה במעבר הראשון, בנוסף שדה foundExtern שאומר אם נמצאה הגדרת extern כלשהי (ולפיה אנו יודעים אם לפתוח את הקובץ במעבר השני), כנל גם לfoundEntry.

היא מכילה finalIC, finalDC, שאומרים מהם הic וdc הסופיים, לשימוש במעבר השני.

על מנת לעשות זאת השתמשנו בפונקציות ובמני הנתונים הבאים:

* int checkFirstCharacter (char\* line, char c)

פונקציה זו, בודקת אם התו הראשון בשורה שהתקבלה מהקובץ הוא תו כלשהו.

הפרמטרים שלה הם:

line – השורה שאנו בודקים

c- התו שאותו אנו בודקים

מחזירה 1 אם c הינו האות הראשונה במחרוזת line (אחרי התעלמות מרווחים) אחרת מחזירה 0

* void fixDataInstruction (char\* line, char parsedLine[MAX\_LINE\_LENGTH \* 2])

פונקציה זו היא פונקצייה שמטפלת בשורה לפני עשיית doData, והיא עושה זאת על ידי הוספת רווחים אחרי פסיקים, כך שstrtok לא ידלג על פסיקים סמוכים.

line – השורה שאותה אנחנו מסדרים

parsedLine – המחרוזת שאלייה אנו נכתוב את השורה המתוקנת, אורכה הוא MAX\_LINE\_LENGTH \* 2, שכן אנו מוסיפים רווחים ולא נרצה לחרוג מהגבלות השורה.

לא מחזירה כלום

* int doData(symbolTable\* table, char \* line, int \*DC, int numberOfLine, symbolTable symbol)

פונקציה זו מעדכנת את מונה הנתונים(DC) בהתאם לכמות המספרים שנמצאו בשורה המכילה את ההוראה .data

הפרמטרים שלה הם:

table – טבלת הסמלים שלנו

line – מחרוזת שמכילה את ההנחיה המלאה שקיבלנו בשורה,

DC - מצביע של מונה הנתונים שלנו

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

symbol - סמל שנוצר במידה והתקבל בשורה

מחזירה0 אם נמצאה שגיאה בשורה נדפיס את השגיאה עם מספר השורה אחרת 1

* int doString(symbolTable\* table, char \* line, int \*DC, int numberOfLine, symbolTable symbol)

פונקציה זו מעדכנת את מונה הנתונים(DC) בהתאם לכמות התווים שנמצאו בשורה המכילה את ההוראה .string

הפרמטרים שלה הם:

table – טבלת הסמלים שלנו

line – מחרוזת שמכילה את ההנחיה המלאה שקיבלנו בשורה

DC - מספר שלם של מונה הנתונים שלנו

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

symbol - סמל במידה והתקבל בשורה

מחזירה: אם נמצאה שגיאה בשורה נדפיס את השגיאה עם מספר השורה ונחזיר 0, אחרת תחזיר 1

* int doStruct(symbolTable\* table, char \* line, int \*DC, int numberOfLine, symbolTable symbol)

פונקציה זו מעדכנת את מונה הנתונים(DC) בהתאם ראשית למספר היחיד אותו היק מקבלת אחר המילה struct בנוסף תוסיף גם כן למונה הנתונים את כמות התווים שנמצאו בשורה המכילה את ההוראה .struct

הפרמטרים שלה הם:

table – טבלת הסמלים שלנו

line – מחרוזת שמכילה את ההנחיה המלאה שקיבלנו בשורה

DC - מספר שלם של מונה הנתונים שלנו

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

symbol - סמל במידה והתקבל בשורה

מחזירה 0 אם נמצאה שגיאה תדפיס את השגיאה ומספר השורה, אחרת תחזיר 1

* int doEntry(symbolTable\* table,char \* line, int \*DC,int numberOfLine,symbolTable symbol)

פונקציה זו בודקת אם הסמל שהתקבל עם ההנחיה הוא סמל חוקי

הפרמטרים שלה הם:

table – טבלת הסמלים שלנו

line – מחרוזת שמכילה את ההנחיה המלאה שקיבלנו בשורה

DC - מספר שלם של מונה הנתונים שלנו

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

symbol - סמל במידה והתקבל בשורה

מחזירה 1 אם הסמל חוקי אחרת מדפיסה שגיאה ואת מספר השורה ומחזירה 0

* int doExtern(symbolTable\* table,char \* line, int \*DC,int numberOfLine, symbolTable symbol)

פונקציה זו בודקת אם הנחיית extern היא לפי הוראות המטלה ופועלת בהתאם להוראות המטלה

הפרמטרים שלה הם:

table – טבלת הסמלים שלנו

line – מחרוזת שמכילה את ההנחיה המלאה שקיבלנו בשורה

DC - מספר שלם של מונה הנתונים שלנו

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

symbol - סמל במידה והתקבל בשורה

מחזירה 1 אם ההנחיה חוקית ולפי תנאי המטלה אחרת מדפיסה שגיאה ואת מספר השורה ומחזירה 0

* int checkValidOperand(char\* operand , int numberOfLine)

פונקציה זו בודקת אם האופרנד התנון חוקי ועומדים בתנאי המטלה

הפרמטרים שלה הם:

operand – מחרוזת האופרנדים שהתקבלה

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

מחזירה 1 האופרנד חוקים ועומד בתנאי המטלה אחרת מדפיסה שגיאה ואת מספר השורה ומחזירה 0

* int isValidCommandSentence(int operandNum, char\* restOfLine , int numberOfLine, char\* firstOperand, char\* secondOperand)

פונקציה זו בודקת אם ההוראה שהתקבלה בשורה היא הוראה חוקית ועומדת בכל תנאי המטלה של הפרויקט

הפרמטרים שלה הם:

operandNum – מספר האופרנדים שהפקודה מקבלת

restOfLine – המחרוזת שמתקבלת אחרי מילת ההוראה

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

firstOperand - מחרוזת המכילה את האופרנד הראשון בהוראה

secondOperand - מחרוזת המכילה את האופרנד השני בהוראה

הפונקצייה תחזיר 1 אם האופרנדים הנוספים של הפקודה תקינים, אחרת תחזיר 0

* int UpdateICforCommandSentence(char\* command, int operandNum , int\* IC, int numberOfLine, char\* firstOperand, char\* secondOperand)

פונקציה זו מעדכנת את מונה ההוראה(IC) אם שורת ההוראה חוקית והפקודה והאופרנדים שלה תואמים את תנאי המטלה

הפרמטרים שלה הם:

Command – מחרוזת שמכילה את ההוראה המלאה שקיבלנו בשורה

operandNum – מספר האופרנדים שהתקבלו עם ההוראה

IC - מצביע של מונה ההוראה שלנו

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

firstOperand - מחרוזת המכילה את האופרנד הראשון בהוראה

secondOperand - מחרוזת המכילה את האופרנד השני בהוראה

מחזירה 0 אם כל התנאים והבדיקות התקבלו על פי תנאי המטלה אחרת מחזירה 1

* int doCommandSentence(char \*command, int \*IC,int numberOfLine,symbolTable symbol)

פונקציה זו מבצעת את שורת ההוראה על פני תנאי המטלה ומבצעת את כל הבדיקות הדרושות לכך בהתאם

הפרמטרים שלה הם:

Command – מחרוזת שמכילה את ההוראה המלאה שקיבלנו בשורה

IC - מצביע של מונה ההוראה שלנו

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

symbol - סמל במידה והתקבל בשורה

מחזירה 1 נמצאה שגיאה בשורת ההוראה אחרת מחזירה 0(בהתאם למה שהפונקצייה UpdateICforCommandSentence מחזירה)

אחד ממבני הנתונים שיש לנו בקובץ זה הוא struct inst  
שמכיל את השדות הבאים:

|  |
| --- |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| const char \*name; |
|  |

|  |
| --- |
| int(\*doInstructions)(symbolTable\* table,char \*line, int \*DC, int numberOfLine,symbolTable symbol); |
|  |

מבנה זה הוא מבנה שמתאים כל שם של instruction , לפונקציות המתאימות לעיל.

(אם נמצא .data נקרא לdoData, וכו)

* int validInstructions(symbolTable\* table,char \*instruction,int \*DC, int numberOfLine,symbolTable symbol)

פונקציה זו בודקת אם שורת ההנחיה שהתקבלה בשורה היא הנחיה חוקית ואם כן מבצעת את הפועלות הדרושות על פי תנאי המטלה לביצוע שורת הנחיה בהתאם לסוג ההנחיה

הפרמטרים שלה הם:

Table- מצביע לראש טבלת הסמלים

instruction – מחרוזת שמכילה את ההוראה שקיבלנו בשורה

DC - מבציע של מונה הנתונים שלנו

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

symbol - סמל במידה והתקבל בשורה

מחזירה 0 אם נמצאה שגיאה בשורת ההוראה (וכמובן שמדפיסה אותה),אחרת 1

* void iCCounter(addressingMode address,addressingMode prevAddress, int \*IC)

פונקציה זו מעדכנת את מונה ההוראות בהתאם לסוג המיעון שהתקבל בשורת ההוראה

הפרמטרים שלה הם:

address – מספר שמייצג את סוג המיעון שהתקבל לאופרנד השני

prevAddress – מספר שמייצג את סוג המיעון של האופרנד הראשון

IC - מצביע של מונה ההוראה שלנו

לא מחזירה דבר

* symbolTable handleLabelDefinition(symbolTable\* table, char\* labelName, flags\* status, int IC, int numberOfLine)

פונקציה זו אחראית על טיפול בהגדרות סמלים חדשים.

היא בודקת שגיאות, ומוסיפה את הסמל החדש לטבלת הסמלים.

הפרמטרים שלה הם:

table – טבלת הסמלים שלנו

labelName – מחרוזת שמייצת את שם הסמל

status – המבנה דגלים שלנו, שבו אנו יודעים אם יש שגיאה

IC - מספר שלם של מונה ההוראה שלנו

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

מחזירה את הסמל שנוצר אם לא נמצאה שגיאה, אחרת מחזירה NULL ומשנה את הerror flag ל1.

* symbolTable createSymbolTable(char\* fileName, flags\* status)

פונקציה זו היא הראשית בקובץ assembleySentence, והיא זו שאחראית על מעבר קובץ am, יצירת טבלת הסמלים וזיהוי שגיאות.

היא מנתחת את המילים שהתקבלו בשורה ומבצעת את הבדיקות הנדרשות בהתאם להנחיות המטלה.

הפרמטרים שלה הם:

fileName – מחרוזת שמו של הקובץ

status – מצביע למבנה המכיל דגלים על מצב האסמבלר.

אם הקובץ לא נפתח תחזיר או נמצאה שגיאה בקובץ תחזיר NULL ותעדכן את status, אחרת תחזיר את טבלת הסמלים שהתקבלה מהקובץ

**SecondPass.c:**

לאחר המעבר הראשון, נבנתה טבלת סמלים ונמצאו רוב השגיאות. מטרתו של קובץ זה זה לקודד את קוד האסמבלי לקבצים ob, ext, ent.

בקובץ זה יש את הפונקציות הבאות:

* void writeToFile(char\* content, FILE\* file, int IC)

פונקציה זו היא אחראית על כתיבה לקובץ ob.

הפרמטרים שלה הם:

Content – מחרוזת המייצגת שני תויים בבייס 32 שיכתבו לקובץ.

File – מצביע לקובץ ob

IC – ערכו של הIC הנוכחי, הוא יכתב גם כן לקובץ.

לא מחזירה כלום.

* void toBase32 (int num, char\* base)

פונקציה זו היא אחראית על המרת מספר לbase32, וכתיבתו למחרוזת מצורפת.

הפרמטרים שלה הם:

num – המספר שאותו אנו ממירים

base – מצביע למחרוזת שבה אנו כותבים את התוצאה המומרת

לא מחזירה כלום.

חילקנו את הקובץ לחלקים, כך שכל חלק אחראי על משהו אחר.

**החלק הבא אחראי על קידוד הוראות הנחייה:**

* void encodeData(symbolTable table, int\* IC, int numberOfLine, FILE\* outFile)

פונקציה זו היא אחראית על קידוד הנחיית data

הפרמטרים שלה הם:

Table – טבלת הסמלים שנבנתה במעבר הראשון

IC – מצביע למונה ההוראות

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

outFile – מצביע לקובץ הdata שבו אנו מקודדים את ההנחיות מידע שלנו.

* void encodeString(symbolTable table, int\* IC, int numberOfLine, FILE\* outFile)

פונקציה זו היא אחראית על קידוד הנחיית string

הפרמטרים שלה הם:

Table – טבלת הסמלים שנבנתה במעבר הראשון

IC – מצביע למונה ההוראות

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

outFile – מצביע לקובץ הdata שבו אנו מקודדים את ההנחיות מידע שלנו.

* void encodeStruct(symbolTable table, int\* IC, int numberOfLine, FILE\* outFile)

פונקציה זו היא אחראית על קידוד הנחיית struct

הפרמטרים שלה הם:

Table – טבלת הסמלים שנבנתה במעבר הראשון

IC – מצביע למונה ההוראות

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

outFile – מצביע לקובץ הdata שבו אנו מקודדים את ההנחיות מידע שלנו.

* void handleEntryAndExtern (char\* firstWord, symbolTable table, int\* IC, int numberOfLine, FILE\* entFile, flags\* status)

פונקציה זו היא אחראית על טיפול בהנחיות entry וextern.

אם זוהי הנחיית entry היא תבדוק שגיאות ותכתוב לקובץ .ent

ואם זוהי הנחיית extern היא תבדוק שגיאות ותכתוב לקובץ .ext

הפרמטרים שלה הם:

firstWord – זוהי סוג ההנחייה שנכתבה, היא תיהיה .extern/.entry

Table – טבלת הסמלים שנבנתה במעבר הראשון

IC – מצביע למונה ההוראות

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

entFile – מצביע לקובץ הent שבו אנו מקודדים הנחיות entity

Status – מצביע למבנה שמכיל את הדגלים שלנו באסמבלר.

* void handleInstructions(symbolTable table, char \*instruction, int \*IC, int numberOfLine, FILE\* outFile);

פונקציה זו היא אחראית על טיפול וקידוד בהנחיות.

הפרמטרים שלה הם:

Table – טבלת הסמלים שנבנתה במעבר הראשון

Instruction – מחרוזת המכילה את סוג ההנחייה

IC – מצביע למונה ההוראות

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

outFile – מצביע לקובץ הdata שבו אנו מקודדים הנחיות מידע

**החלק הבא אחראי על קידוד פקודות:**

* int encodeNoneOperandsCommand(symbolTable table, char \*command, int \*IC,int numberOfLine, FILE\* outFile)

פונקציה זו היא אחראית על טיפול וקידוד בפקודות בעלות 0 אופרנדים.

הפרמטרים שלה הם:

Table – טבלת הסמלים שנבנתה במעבר הראשון

command – מחרוזת המכילה את הפקודה בשורה

IC – מצביע למונה ההוראות

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

outFile – מצביע לקובץ הcmd שבו אנו מקודדים הנחיות הוראה

* int encodeOneOperandsCommand (symbolTable table, char \*command, int \*IC,int numberOfLine, FILE\* outFile, FILE\* extFile)

פונקציה זו היא אחראית על טיפול וקידוד בפקודות בעלות אופרנד יחיד.

הפרמטרים שלה הם:

Table – טבלת הסמלים שנבנתה במעבר הראשון

command – מחרוזת המכילה את הפקודה בשורה

IC – מצביע למונה ההוראות

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

outFile – מצביע לקובץ הcmd שבו אנו מקודדים הנחיות הוראה

extFile - מצביע לקובץ ext שבו אנו כותבים הופעות של externals.

* int encodeTwoOperandsCommand (symbolTable table, char \*command, int \*IC,int numberOfLine, FILE\* outFile, FILE\* extFile)

פונקציה זו היא אחראית על טיפול וקידוד בפקודות בעלות שני אופרנדים.

הפרמטרים שלה הם:

Table – טבלת הסמלים שנבנתה במעבר הראשון

command – מחרוזת המכילה את הפקודה בשורה

IC – מצביע למונה ההוראות

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

outFile – מצביע לקובץ הcmd שבו אנו מקודדים הנחיות הוראה

extFile - מצביע לקובץ ext שבו אנו כותבים הופעות של externals.

* int encodeCommandSentence (symblTable table, char\* command, int\* IC, int numberOfLine, FILE\* outFile, FILE\* extFile);

פונקציה זו היא אחראית על טיפול וקידוד בשורות פקודה

הפרמטרים שלה הם:

Table – טבלת הסמלים שנבנתה במעבר הראשון

command – מחרוזת המכילה את הפקודה

IC – מצביע למונה הנתונים

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

outFile – מצביע לקובץ הcmd שבו אנו מקודדים פקודות

extFile – מצביע לקובץ ה.ext שבו אנו מקודדים מופעים של externals

- מצביע לקובץ ext שבו אנו כותבים הופעות של externals.

**החלק הבא אחראי על קידוד אופרנדים על פי שיטת המיעון שלהם:**

* int handleImmediateAddress(char\* operand, int\* IC, int numberOfLine, FILE\* outFile);

פונקציה זו היא אחראית על טיפול וקידוד באופרנדים בעלי מיעון מיידי.

הפרמטרים שלה הם:

operand – מחרוזת המכילה את האופרנד

IC – מצביע למונה הנתונים

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

outFile – מצביע לקובץ הcmd שבו אנו מקודדים פקודות

* int handleDirectAddress (symblTable table, char\* operand, int\* IC, int numberOfLine, FILE\* outFile, FILE\* extFile);

פונקציה זו היא אחראית על טיפול וקידוד באופרנדים בעלי מיעון ישיר.

הפרמטרים שלה הם:

Table – טבלת הסמלים שנבנתה במעבר הראשון

operand – מחרוזת המכילה את האופרנד

IC – מצביע למונה הנתונים

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

outFile – מצביע לקובץ הcmd שבו אנו מקודדים פקודות

extFile – מצביע לקובץ ה.ext שבו אנו מקודדים מופעים של externals

* int handleAddressAccess (symblTable table, char\* operand, int\* IC, int numberOfLine, FILE\* outFile, FILE\* extFile);

פונקציה זו היא אחראית על טיפול וקידוד באופרנדים בעלי מיעון גישה לרשומה.

הפרמטרים שלה הם:

Table – טבלת הסמלים שנבנתה במעבר הראשון

operand – מחרוזת המכילה את האופרנד

IC – מצביע למונה הנתונים

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

outFile – מצביע לקובץ הcmd שבו אנו מקודדים פקודות

extFile – מצביע לקובץ ה.ext שבו אנו מקודדים מופעים של externals

* int handleAddressingAccesses (addressingMode operandMode, symblTable table, int isDest, char\* operand, int\* IC, int numberOfLine, FILE\* outFile, FILE\* extFile);

פונקציה זו היא אחראית על טיפול וקידוד מילות מידע נוספות, על ידי קריאות לפונקציות המתאימות על פי שיטות המיעון של האופרנדים.

הפרמטרים שהיא מקבלת הם:

operandMode – שיטת המיעון של האופרנד

Table – טבלת הסמלים שנבנתה במעבר הראשון

isDest – משתנה בוליאני שאומר אם האופרנד שנשלח הוא מקור או יעד

operand – מחרוזת המכילה את האופרנד

IC – מצביע למונה הנתונים

numberOfLine – מספר השורה הנוכחי בקובץ הקלט

outFile – מצביע לקובץ הcmd שבו אנו מקודדים פקודות

extFile – מצביע לקובץ ה.ext שבו אנו מקודדים מופעים של externals

**פונקציות כלליות של המעבר השני:**

* void handleFinalOutputFiles (char\* fileName, FILE\* cmdFile, FILE\* dataFile, flags\* status);

פונקציה זו היא אחראית על שילוב שתי קבצי ה.cmd ו.data לקובץ .obj המכיל בהתחלה את הפקודות ולאחר מכן את הנתונים.

הפרמטרים שהיא מקבלת הם:

filename – שם הקובץ שעליו עוברים

cmdFile – מצביע לקובץ הקידוד של הפקודות

Datafile – מצביע לקובץ הקידוד של המידע

Status – מבנה המכיל דגלים שבהם יש מידע על מצב האסמבלר

* openFiles(char\* fileName, FILE\*\* inputFile, FILE\*\* cmdFile, FILE\*\* dataFile, FILE\*\* entFile, FILE\*\* extFile, flags\* status)

פונקציה זו היא אחראית על פתיחת כל הקבצים המשמשים אותנו במעבר השני.

הפונקצייה מקבלת מצביעים למצביעים של קבצים, שכן היא צריכה לשנות את המשתנים.

הפרמטרים שהיא מקבלת הם:

fileName – שם הקובץ שעליו עוברים

inputFile – מצביע של הקובץ קלט am

cmdFile – מצביע של קובץ הקידוד של הפקודות

dataFile – מצביע לקובץ הקידוד של המידע

entFile – מצביע לקובץ המכיל קידוד של תוויות entities

extFile – מצביע לקובץ המכיל קידוד של תוויות externals

Status – מבנה המכיל דגלים שבהם יש מידע על מצב האסמבלר

* encodeAssembly(char\* fileName, symbolTable table, flags\* status)

פונקציה זו היא הראשית בקובץ secondPass, והיא אחראית על המעבר של קובץ הקלט, וקידודו.

הפרמטרים שהיא מקבלת הם:

fileName – שם הקובץ שעליו עוברים

Table – טבלת הסמלים שנבנתה במעבר הראשון

Status – מבנה המכיל דגלים שבהם יש מידע על מצב האסמבלר