**מבנה המחשב – פרוייקט ISA, דוקומנטציה**

איתי מרקמן – 206900193

עידו גפן

בן דב בלוך

**רקע קצר**

בפרוייקט אנו נדרשים לחקור את עבודתו של מעבר בשם SIMP הדומה למאבד MIPS ולראות איך הוא מריץ תוכנות הכתובות בשפת האסמבלי הייחודית שלו.

בשפה זו ישנם 19 פקודות אסמבלי המוצגות להלן, כאשר בקובץ הזיכרון(memin.txt, memout.txt) כל פקודה רשומה כשורה הקסאדצימלית ברוחב 8 תווים. או 32 ביטים. הפקודה בנויה ככה מימין לשמאל:

תווים 1:3 – שדה ה-Immediate שיוכנס לתוך אוגר ה-Immediate בביצוע הפעולה

תו 4 – אוגר rt

תו 5 – אוגר rs

תו 6 – אוגר rt

תווים 7-8 –opcode, שם הפקודה

להלן טבלה עם כל הפקודות שתוכנות האסמבלר והסימולטור תומכות בהן:

|  |  |  |
| --- | --- | --- |
| מספר opcode בהקסאדצימלי | שם בשפת אסמבלי | משמעות |
| 00 | Add |  |
| 01 | Sub |  |
| 02 | And |  |
| 03 | or |  |
| 04 | Sll |  |
| 05 | Sra | הזזה אריתמטית עם הארכת סימן |
| 06 | Srl | *הזזה לוגית* |
| 07 | beq |  |
| 08 | bne |  |
| 09 | blt |  |
| 0A | bgt |  |
| 0B | Ble |  |
| 0C | bge |  |
| 0D | jal |  |
| 0E | lw |  |
| 0F | sw |  |
| 11 | reti |  |
| 12 | in |  |
| 13 | out |  |
| 14 | halt | Halt execution – exit simulator |

בכל פקודות הbranch. ההתניה נעשית רק על 11 הביטים הימניים ביותר של האוגרים. ו-PC פירושו מספר השורה הנוכחית וmem זה כתובת של שורה בזיכרון.

הוראות בקבצי אסמבלי מתחילות ב# וישנה אפשרות להכניס מספר של תא בזיכרון כlabel. כלומר לתת לו כינוי ואז לעשות נקודותיים. כך אפשר לקפוץ ממקום למקום בקוד יותר בקלות

פקודת אסמבלי אחרונה היא הפקודה ".word" פקודה זו מאחסנת מילה של 32 ביטים(8 תווים הקסאדצימלים) ישירות בזיכרון. היא בנויה כך:

.word address data

כמו כן למעבד 16 אוגרים עם תפקידים שונים

|  |  |  |
| --- | --- | --- |
| מספר(הקסאדצימלי) | שם | תפקיד |
| 0 |  | תמיד מכיל אפס |
| 1 |  | הפניה לשדה ה-immediate של הפקודה |
| 2 |  | ערך מוחזר מפונקציה |
| 3 |  | ארגומנט של פונקציה |
| 4 |  | ארגומנט של פונקציה |
| 5 |  | משתנה זמני |
| 6 |  | משתנה זמני |
| 7 |  | משתנה זמני |
| 8 |  | משתנה זמני |
| 9 |  | משתנה זמני נשמר |
| A |  | משתנה זמני נשמר |
| B |  | משתנה זמני נשמר |
| C |  | Global pointer(static data) |
| D |  | מצביע למחסנית |
| E |  | Frame pointer |
| F |  | כתובת חזרה לאחר שימוש בפונקציה |

אנו מניחים שתדר המעבד הוא 256 הרץ ושכל פקודה לוקחת מחזור שעון אחד

**האסמבלר –Assembler.c**

תפקיד תוכנת האסמבלר היא לקחת קובץ asm הכולל בתוכו תוכנית אסמבלי בשפה שתוארה בפרק הקודם. ולהמירו לקובץ txt המכיל שפת מכונה הקסאדצימלית המתארת את תמונת הזיכרון לפי ריצת תכנה. האסמבלר בנוי כך שיוכל לתמוך בתוכנות גדולות ככל האפשר(אינסופיות בתאוריה) אך מניח כי לייבל מוכב מ50 תווים. אורך שורת קוד הוא עד 500 תווים, מספר immediate הוא עד 50 תווים, ושם אוגר/אופקוד עד 6 תווים וכתוב בשפה c מתחילתו ועד סופו. ישנו משתנה גלובלי MAX\_LINE המוגדר לעד 500 תווים

**הערה לבודק**– האסמבלר מסתמך על הספריות stdio.h,stdlib.h,string.h ולכן יש לוודא שהן מותקנות לפי הרצה. השימוש בספריות אלו חוקי לפי הוראות הפרוייקט

מבנה עזר –label:

על מנת לתמוך בתוכנות בגודל אינסופי, האסמבלר מאחסן מידע על הקוד שהוא קורא וממיר ברשימות מקושרות הבנויות ממבנים בשפת סי. בכל מבנה מידע על האיבר הנוכחי ומצביע לבא ברשימה. מתחת לכל הגדרה של מבנה בקוד ישנן פונקציות עזר הקשורות אליו. מבנה קוד זה הוא הדבר הקרוב ביותר ל"קוד מונחה עצמים" שניתן לבנות בשפת סי

נסתכל על המבנה label:

// the label linked list will help us store info about the labels. it will be built at the first iteration and used and the second. then destroyed

typedefstructLabel

{

// name will store the name of the label

char name[50];

// location will store the line number and will be the immediate value in related jump and branch commands

int location;

// a pointer to the next label

structLabel\* next;

} label;

הפונקציה המגדירה את המבנה, שכולל מחרוזת nameעד 50 תווים עם שם הלייבל, מספר locationהמתאר את מיקומו בזיכרון(השורה שיקפצו אליה) ומצביע nextללייבל הבא ברשימה

// this function creates a label from the given name and location

label\* create\_label(charname[50], intlocation)

{

// allocate memory for the label and create a pointer to it

label\* new\_label = (label\*)malloc(sizeof(label));

// if allocation successful. insert data to label

if (new\_label != NULL) {

// use strcpy to insert a string

strcpy(new\_label->name,name);

// the other insertions are easy

new\_label->location = location;

new\_label->next = NULL;

}

return new\_label;

}

ה"constructor" של מבנה זה. לוקח מחרוזת ומבנה, מקצה זיכרון, ואם היא מצליחה להקצות, שמה את מה שהיא קבלה בתאים הרלוונטים ומחזירה מצביע לlabel החדש

// adds a label to the front of an existing label list with given name and location.

// will be used in the first iteration

label\* add(label\* head, charname[50], intlocation) {

// build the label and check for success. otherwise return a null

label\* new\_label = create\_label(name, location);\_ if (new\_label == NULL)

returnNULL;

// the next pointer will point to the original head

new\_label->next = head;

return new\_label;

}

פונקציה זו מקבלת את שם הלייבל ומיקומו ומוסיפה לייבל חדש בתחילת הרשימה.

// this will scan the label list "head" and look for "name". it will return it's location.

// this will be used in the second iteration

int find(label\* head, charname[50]) {

// current - the current label's name

char current[50];

// start with the head

strcpy(current, head->name);

// strcmp returns 0 if names are equal

while (strcmp(current, name) != 0) {

// go to next label

head = head->next;

// in case not found - return -1

if (head == NULL) {

return -1;

}

// update name to current name

strcpy(current, head->name);

}

// return the current's location

returnhead->location;

}

פונקציה זו נקראת בין האיטרציה השנייה של האסמבלר לכתיבת תמונת הזיכרון. היא מקבלת שם של לייבל ומצביע לרשימה. ועוברת על הרשימה איבר איבר(המצביע current מצביע לאיבר הנוכחי) עד שהיא מוצאת את הלייבל שהאסמבלר רוצה לחפש ומחזירה את מיקומו בזיכרון. אם הלייבל לא נמצא מוחזר -1

// destroys the list and frees the memory

void destroy(label\* head)

{

// temp - a pointer to a label we are going to destroy after updating head

label\* temp;

// all the way to the end

while (head != NULL)

{

// temp gets the current node while head advances

temp = head;

head = head->next;

// we destroy temp and free the memory

free(temp);

}

}

פונקציה הנקראת בסוף הקוד שהורסת את הלייבל ומשחררת את הזיכרון.המשתנה temp עוזר בשחרור כך שתמיד יהיה לנו מצביע לראש הרשימה הנוכחי

מבנה עזר - שורת זיכרון MemoryLine:

מבנה זה מתאר שורה בזיכרון. ובו מחרוזות opcode, rd, rs, rt,imm המתארים כל שדה של הפקודה באסמבלי, מספר pos המתאר את מספר השורה בזיכרון אליה צריך לפנות ומצביע next לאיבר הבא ברשימה. כמו כן, אם השורה מתארתפקודת .word כל השדות כוללים את הערך NONO והשדה imm את מילת הזיכרון שיש לאחסן

// this struct will be used to save the memory lines

typedefstructMemoryLine {

// the opcode of the command

char opcode[6];

// the registers

char rd[6];

char rs[6];

char rt[6];

// immediate value

char imm[50];

// position of character in line

int pos;

// memory will also be a linked list to support infinite length programs

structMemoryLine\* next;

}MemoryLine;

הגדרה המבנה

גם למבנה זה ישנן פונקציות נלוות:

// creates new memory line with no "next"

MemoryLine\* create\_line(charopcode[6], charrd[6], charrs[6], charrt[6], charimm[50], intpos) {

// allocate memory for the label and create a pointer to it

MemoryLine\* new\_line = (MemoryLine\*)malloc(sizeof(MemoryLine));

// if allocation successful. insert data to label

if (new\_line != NULL) {

// use strcpy to insert the strings

strcpy(new\_line->opcode, opcode);

strcpy(new\_line->rd, rd);

strcpy(new\_line->rs, rs);

strcpy(new\_line->rt, rt);

strcpy(new\_line->imm, imm);

new\_line->pos = pos;

// no next defined

new\_line->next = NULL;

}

return new\_line;

}

ה"constructor" של המבנה, מקבל את ערכי כל השדות, מקצה זיכרון ואם הוא מצליח מציב את הערכים שהוא קיבל כך שהאיבר הבא ברשימה לא מוגדר

// adds line to memory structure. this line will be added to the end to let the writing run it like an array

MemoryLine\* add\_line(MemoryLine\* head, charopcode[6], charrd[6], charrs[6], charrt[6], charimm[50], intpos)

{

// the last line as for now

MemoryLine\* tail;

// create a line

MemoryLine\* new\_line = create\_line(opcode, rd, rs, rt, imm, pos);

// if the new line is null. do nothing

if (new\_line == NULL)

returnNULL;

// and return the current if no head supplied

if (head == NULL)

return new\_line;

// get the "tail" to the end of the list

tail = head;

while (tail->next != NULL)

tail = tail->next;

// add the new line

tail->next = new\_line;

// return updated memory

returnhead;

}

פונקציה שמוסיפה שורת זיכרון נוספת לרשימה. היא משתמשת במצביע tail על מנת להוסיף את השורה בסוף הרשימה. דבר שחשבו שייקל על התוכנה בהמשך.

// destroy the memory line list and free the memory the assembler used

void destroy\_memLine(MemoryLine\* head) {

// temp - a pointer to a line we are going to destroy after updating head

label\* temp;

// all the way to the end

while (head != NULL)

{

// temp gets the current node while head advances

temp = head;

head = head->next;

// we destroy temp and free the memory

free(temp);

}

}

דומה מאוד ל-destroy של הלייבלים. פונקציה זו לוקחת רשימה של שורות זיכרון, ומשחררת אותה מהזיכרון(של האסמבלר). המשתנה temp עוזר בשחרור כך שתמיד יהיה לנו מצביע לראש הרשימה הנוכחי

// get the memory line at position. can return null if does not exist

MemoryLine\* getAtPos(MemoryLine\* head, intpos) {

// go until you find

while (head != NULL&&head->pos != pos)

head = head->next;\_ returnhead;

}

פונקציה פשוטה מאוד שמקבלת מספר של שורת זיכרון ומחזירה מצביע לאיבר הרשימה שכולל את המידע עליה

מבנהעזר - זיכרון –Memory:

שדה זה כולל רשימה headשל מידע על שורות הזיכרון. ומספר last המכיל את מספר השורה האחרונה בזיכרון. מספר lastזה יהיה שווה למספר שורות הקוד שאינן הערה או לייבל במקרה שאין פקודות .word במקרה ופקודות כאלו קיימות הוא יתאר את הכתובת של האחרונה או הכתובת שורת הקוד האחרונה, מה שיותר גדול

// Memory struct and related functions. it is used so the second iteration can return two values.

typedefstructMemory

{

// head of memory line list

MemoryLine\* head;

// position of last

int last;

}Memory;

למבנה זה רק פונקציה אחת. שכן השימוש בפקודות שרלוונטיות אליו מועט. והיא הפונקציה הבאה שמשחררת את הזיכרון של המבנה על ידי שחרור הרשימה

// destroys the memory struct after use

void destroy\_mem(Memory\* mem) {

// destroy the memory's line list

destroy\_memLine(mem->head);

// free the memory object's own memory

free(mem);

}

איטרציות על הקבצים:

האסמבלר מבצע את עבודתו באיטרציות על הקבצים. שתי איטרציות על קובץ הקלט. אחת בה הוא משיג מידע על הלייבלים, ואחת בה הוא משיג מידע על שורות הזיכרון. אחרי שתי איטרציות אלו הוא מחליף את שמות הלייבלים היכן שצריך(בעיקר הוראות קפיצה) במספרי כתובות הזיכרון שלהם ורושם את תמונת הזיכרון בקובץ הפלט

איטרציה ראשונה – יצירת רשימת הלייבלים:

איטרציה זו עוברת על הקוד(אותו היא מקבלת במצביע asembl לקובץ הקלט). בודקת אם שורת קוד כוללת לייבל על ידי זיהוי התו ":" ואם כן מאחסנת מידע על הלייבל ברשימת הלייבלים

// the code of the first iteration. goes trough the file row by row and looks for labels, then adds them to the label list

label\* createLabelList(FILE \*asembl) {

הגדרות משתנים אשר נעשה בהם שימוש. הערת הקוד מעל כל משתנה מסבירה על תפקידו

// the code row's index. where the PC will go after reading the label

int rowIndex = 0;

// line the current line being read, tav1 is the first char and i used to check for remarks

char line[MAX\_LINE], tav1;

// tav - current char when reading label name

char tav;

// will contain the name of the label once iteration is complete

char lable\_line[50];

// the dots are used to say "this is a label"

char dots[50];

strcpy(dots, ":");

// k is the char index for label name read, j is the index in the label name string we are building

int k, j;

// option will determine if it's a label only line or a label + command line

int option;

// counter will be the line number in the new hexadecimal code. it will go up when a line that gets translated is found

int counter = 0;

// the label list's head

label\* head = NULL;

אחרי שסיימנו עם ההגדרות. נשתמש בלולאה while על מנת לקרוא את קובץ הקלט ולאתר לייבלים. הלולאה מסתיימת כאשר מתרחש אירוע feof של המצביע לקובץ

// go all the way trough the file

while (!feof(asembl)) {

דבר ראשון – קריאת השורה

// read a command from the assembler file

fgets(line, MAX\_LINE, asembl);

// reset option

option = 0;

איתור סוג השורה. במקרה של שורה ריקה או פקודת .word או הערה בתו הראשון עוברים לשורה הבאה

if (strcmp(line, "\n") == 0) //If line is blank, continue

continue;

tav1 = line[0];

if (tav1 == '#') //If line is Remark, continue

continue;

if (strstr(line, ".word") != NULL) //If line is .word, continue

continue;

אך אם אותר התו נקודותיים האסמבלר מבין שהוא הגיע לשורת לייבל

if (strstr(line, dots) != NULL) //If dots are found, this is a label

{

אך לא לפני שהוא עורך בדיקה אם הנקודותיים נמצאות בתוך הערה מעבר לתו # ואז זו אזעקת שווא ועוברים לשורה הברה

if (strstr(line, "#") != NULL) // however, ":" can be in a remark. so check for that as well, if so go to another line

if ((strstr(line, dots)) > (strstr(line, "#")))

continue;

קוראים את שם הלייבל. הקריאה עוברת על הקוד תו תו עד שהיא מגיעה לתו ":" ומתעתיקה את שם לייבל למחרוזת lable\_line

//Read the label name, first reset indexes

k = 0;

j = 0;

do {

// get current char

tav = line[k];

// two dots is where it ends so skip

if (tav != ':') {

// don't read tabs and spaces

if (tav != '\t')

if (tav != ' ')

{

// grab the read char and put it in name string

lable\_line[j] = tav;

// increment name string index

j++;

}

// increment reading index

k++;

}

} while (tav != ':');

בסוף. הלייבל צריך להיגמר בתו null כמו כל מחרוזת בסי

// label name is null terminated

lable\_line[j] = '\0';

חלק הקוד הבא מקדם את האינדקס בשורת קוד האסמבלי k על מנת לדעת אם שורת הזיכרון עליה הלייבל מצביע מתפרשת בקוד על שתי שורות קוד(לייבל ופקודה) או על שורה אחת שכוללת גם לייבל וגם פקודה על מנת שנוכל לקדם את אינדקס שורת הזיכרון counter בהתאם

k++; // Check if the line is lable line only by seeing if there are only spaces and tabs till the end

while ((line[k] == ' ') || (line[k] == '\t'))

k++;

// option is 1 on label only line, otherwise 0

if ((line[k] == '\n') || (line[k] == '#'))

option = 1;

אחרי הבדיקה. אנו מוסיפים את הלייבל לרשימת הלייבלים

// finally we add the label to label list

head = add(head, lable\_line, counter);

ובמקרה של לייבל בלבד. מחזירים את אינדקס כתובת הזיכרון אחד אחורה על מנת לא לקדם אותו בשורת הזיכרון

if (option == 1) { // Only label line - add label and decrement counter

counter = counter - 1;

}

}

לאחר ההצבה. נקדם את האינדקס k למיקום בו מתחילה הפקודה הבאה בתור הכנה אליה

k = 0; // Check if the current line is space line using k - most commands in fib.asm and our files start with a tab or a space

if ((line[k] == '\t') || (line[k] == ' '))

k++;

if (line[k] == '\n')

continue;

קידום אינדקס שורת הזיכרון

// increment hexa file line counter

counter++;

}

בסוף, נחזיר את רשימת הלייבלים

// return the list

return head;

}

איטרציה שניה – רשימת שורות קובץ הזיכרון:

איטרציה זו גם עוברת על קוד האסמבלי, אך היא בודקת עבור כל שורה בקובץ איך היא משפיעה על השורה בקובץ הזיכרון. ומכניסה את המידע לרשימת שורות הזיכרון. המצביע file הינו מצביע לקובץ הקלט

// the second iteration. returns the memory list

Memory\* SecondRun(FILE\* file) {

הגדרת משתנים בהם נעשה שימוש. מעל כל משתנה ישנה הערה המייצגת את תפקיד המשתנה

// k is the index of the current char being read

int k = 0;

// i is the current position in the file

int i = 0;

// pos1 is the last line of the memory file

int pos1 = 0;

// pos is the address in .word commands, as an int

int pos = 0;

// j saves the index in the word for the copy process

int j = 0;

המחרוזות line תאחסן את השורה הנוכחית בקוד, tav את התו הנוכחי שאנו קוראים ושאר המחרוזות פה הן פירוק של הפקודה

// line will house the current line. tav is the character we are currently reading and option, rd, rs, rt and imm are the command's values

char line[MAX\_LINE], tav, option[6], rd[6], rs[6], rt[6], imm[50];

אלו מטפלות במקרה של פקודת .word כאשר wordP הוא הכתובת וwordN הוא המילה שאנו מכניסים לזיכרון

// wordP houses the address and wordN houses the data in case of a word command

char wordP[15], wordN[15];

// used to detect labels. because something might be past them

char \*dots = ":";

// used to detect the special ".word" command

char wo[6] = ".word";

// the Memory list's head

MemoryLine\* head = NULL;

// tav1 will save the first character of the line

char tav1;

לולאה זו קוראת את הקובץ שורה שורה ומעבירה וכותבת תוך כדי את רשימת שורות הזיכרון ומעדכנת את מספר השורה האחרונה בקוד. בהגעה לשורה null הקוד נעצר. כמו כן הלולאה בנויה בצורה שמונעת מהקוד להיקרא פעמיים ומעתיקה את השורה הנוכחית מהקובץ למשתנה line תוך כדי ההתניה

// the loop reads the file line by line. and upon reaching null it stops as that's where the file ends

while ((fgets(line, MAX\_LINE, file)) != NULL) // To prevent last line reading twice

{

כאן בודקים את סוג השורה. במקרה של שורה ריקה עוברים לשורה הבאה

if (strcmp(line, "\n") == 0) ///////// in case of a Blank line, go

continue;

אחרת הבדיקה נעשית על פי התו הראשון בשורה. אם זו הערה עוברים שורה

// get first line

tav1 = line[0];

if (tav1 == '#') ///////// in case of a Remark line, go

continue;

במקרה שהשורה היא .word מעתיקים את הכתובת והמילה בזיכרון

if (strstr(line, wo) != NULL) ///////// in case of the special .word order

{

מעבירים את אינדקס התו הנוכחי מעבר לכל הרווחים. שמה מתחילה הכתובת

// reset k index

k = 0;

// go past all the spaces

while (line[k] != ' ')

k++;

k++;

כעת מעתיקים את הכתובת תו תו עד שהיא מסתיימת בתו רווח

j = 0; //Copy Address. first reset j then copy char char until the next space

while (line[k] != ' ')

{

wordP[j] = line[k];

j++;

k++;

}

לא לשכוח לסיים את המחרוזת בnull

// terminate string with null and increment end to next char

wordP[j] = '\0';

k++;

כעת נעתיק את מילת הזיכרון באותה צורה.

j = 0; //Copy Data. using the same way.

while (line[k] != ' ')

{

// but detect an end of line string because after the data there can be a line end

if (line[k] == '\n')

break;

wordN[j] = line[k];

j++;

k++;

}

wordN[j] = '\0';

פקודות .word מבוצעות כבר בשלב האסמבלר. ולכן אנו נעתיק את שורת הזיכרון שאליה אנו כותבים את המילה למספר שורת הזיכרון באיבר הרשימה הנוכחי. ובמקרה של כתובת הקסאדצימלית נמיר לכתובת דצימלית

if (wordP[0] == '0') // change Address int. the if block considers an hexadecimal input

{

if (wordP[1] == 'x')

pos = strtol(wordP, NULL, 16);

}

else// and the else blocks considers a decimal input

{

pos = atoi(wordP);

}

עבור .word כל השדות הינם NONO ושדה ה-immediate מכיל את המילה שאנו רוצים להטמיע בזיכרון

// now. we will save the command in the memory list. NONO will be used as an indicator when writing to turn the command into a .word

strcpy(option, "NONO");

strcpy(rd, "NONO");

strcpy(rs, "NONO");

strcpy(rt, "NONO");

// immediate - data

strcpy(imm, wordN);

לבסוף נוסיף את המידע על השורה כאיבר ברשימה. כאשר השורה בזיכרון היא הכתובת address

// save line to line list

head = add\_line(head, option, rd, rs, rt, imm, pos);

ונעדכן את מיקום השורה האחרונה בקובץ(המשתנה pos1).

// update the location of the end of the memory

if (pos > pos1)

pos1 = pos;

}

במקרה של פקדה רגילה ולייבל.

elseif (strstr(line, dots) != NULL)//in case of regular order and label

{

והערה

if (strstr(line, "#") != NULL)//now we check if the dots is remark and not a label

{

if ((strstr(line, dots)) < (strstr(line, "#")))//label line- check if the line include only label or order

{

התחל בדיקה אחרי הנקודותיים של הלייבל

// check if the line includs only lable or #

k = 0;

while (line[k] != ':')

k++;

k++;

ואם זה רק לייבל. רד שורה. אחרת לך לתחילת הפקודה

if (line[k] == '\n')

continue;

else

while ((line[k] == ' ') || (line[k] =='\t'))

k++;

ורד שורה אם זה לייבל שאחריו רווחים או לייבל והערה בלבד

if (line[k] == '\n')

continue;

if (line[k] == '#')

continue;

אם הגענו לפה, הבנו שזה המקרה של לייבל ופקודה. נתחיל בקריאת הopcode(עד הרווח)

// lable line and order

j = 0; // Read the order

while (line[k] != ' ')

{

option[j] = line[k];

j++;

k++;

}

ונסגור את המחרוזת שלו בנאל

option[j] = '\0';

נקרא את שם האוגר rd עד הפסיק שמפריד בין שמות האוגרים

k++;

j = 0; // Read rd

while (line[k] != ',')

{

rd[j] = line[k];

j++;

k++;

}

ונסיים בנאל, ואז נתקדם שני תווים כי יש גם פסיק וגם רווח

rd[j] = '\0';

k = k + 2;

כעת נעשה אותו דבר לקריאת האוגר rs

j = 0; // Read rs

while (line[k] != ',')

{

rs[j] = line[k];

j++;

k++;

}

rs[j] = '\0';

k = k + 2;

והאוגר rt

j = 0; // Read rt

while (line[k] != ',')

{

rt[j] = line[k];

j++;

k++;

}

rt[j] = '\0';

k = k + 2;

לבסוף, נקרא את ערך הimmediate. כאשר הוא יכול להסתיים בטאב, רווח, הערה או שורה חדשה

j = 0; // Read immidiate

while (line[k] != ' ')

{

if ((line[k] == '\t') || (line[k] == '#') || (line[k] == '\n'))

break;

imm[j] = line[k];

j++;

k++;

}

imm[j] = '\0';

נשמור את המחרוזות שקראנו לרשימת שורות הזיכרון ב-i יש את אינדקס שורת הקוד. האסמבלר מניח שהזיכרון מתחיל בקוד וכל ה.word-ים אחרי הקוד. Pos1 כאמור שורת הזיכרון האחרונה

// save line to line list

head = add\_line(head, option, rd, rs, rt, imm, i);

נקדם את אינדקס שורת הקוד הנוכחית של

i++;

if (i > pos1) //Update last line position

pos1 = i;

}

חלק זה של הקוד מתאר את המקרה שאין לייבל אבל יש הערה שבתוכה התו נקודותיים

else// The line has no lable

{

נקרא את חלקי הפקודה כמו בחלק הקודם של הקוד

j = 0; // Read the order

while (line[k] != ' ')

{

option[j] = line[k];

j++;

k++;

}

option[j] = '\0';

k++;

j = 0; // Read rd

while (line[k] != ',')

{

rd[j] = line[k];

j++;

k++;

}

rd[j] = '\0';

k = k + 2;

j = 0; // Read rs

while (line[k] != ',')

{

rs[j] = line[k];

j++;

k++;

}

rs[j] = '\0';

k = k + 2;

j = 0; // Read rt

while (line[k] != ',')

{

rt[j] = line[k];

j++;

k++;

}

rt[j] = '\0';

k = k + 2;

j = 0; // Read immidiate

while (line[k] != ' ')

{

if ((line[k] == '\t') || (line[k] == '#') || (line[k] == '\n'))

break;

imm[j] = line[k];

j++;

k++;

}

imm[j] = '\0';

ונשמור ונקדם את השורה הנוכחית של חלק הקוד של הזיכרון(i) והאחרונה(pos1)

// save line to line list

head = add\_line(head, option, rd, rs, rt, imm, i);

i++;

if (i > pos1) //Update last line position

pos1 = i;

}

}

חלק זה של הקוד בודק את המקרה בו יש לייבל ואין הערה

else//There is no remark line

{

הולכין לאחרי הלייבל

k = 0;

while (line[k] != ':')

k++;

עוד תו כי לא ספרנו את הנקודותיים

k++;

אם זה רק לייבל, עוברים שורה

if (line[k] == '\n')

continue;

else

ואז קופצים על כל הרווחים והטאבים עד שמגיעים למקרה של הערה שלא נתפסה בבדיקה או שורה חדשה

while ((line[k] == ' ') || (line[k] == '\t'))

k++;

if (line[k] == '\n')

continue;

if (line[k] == '#')

continue;

הגעת לפה, הגעת לפקודה, נעתיק כמו מקודם את כל חלקי הפקודה למחרוזות הרלוונטיות

// lable line and order

j = 0; // Read the order

while (line[k] != ' ')

{

option[j] = line[k];

j++;

k++;

}

option[j] = '\0';

k++;

j = 0; // Read rd

while (line[k] != ',')

{

rd[j] = line[k];

j++;

k++;

}

rd[j] = '\0';

k = k + 2;

j = 0; // Read rs

while (line[k] != ',')

{

rs[j] = line[k];

j++;

k++;

}

rs[j] = '\0';

k = k + 2;

j = 0; // Read rt

while (line[k] != ',')

{

rt[j] = line[k];

j++;

k++;

}

rt[j] = '\0';

k = k + 2;

j = 0; // Read immidiate

while (line[k] != ' ')

{

if ((line[k] == '\t') || (line[k] == '#') || (line[k] == '\n'))

break;

imm[j] = line[k];

j++;

k++;

}

imm[j] = '\0';

ואחרי שהכל הועתק שומרים ברשימה

// Copy Line to MemoryLine structure

head = add\_line(head, option, rd, rs, rt, imm, i);

i++;

if (i > pos1) //Update last line position

pos1 = i;

}

}

לבסוף, נטפל במקרה הפשוט ביותר. פקודה בלי הערה ובלי לייבל.

else// Order line only

{

לך לאחרי הרווחים והטאבים

k = 0;

while ((line[k] == ' ') || (line[k] == '\t')) // roll to end of spaces

k++;

ואם בכל זאת יש הערה או שורה חדשה(בדיקות קודמות לא הצליחו) המשך

if (line[k] == '#')

continue;

if (line[k] == '\n')

continue;

j = 0;

העתק את האופקוד עד התו $

do {

tav = line[k];

if (tav != '$')

{

if (tav != '\t')

if (tav != ' ')

{

option[j] = tav;

j++;

}

k++;

}

} while (tav != '$');

option[j] = '\0';

ואת האוגרים

j = 0; // Read rd

while (line[k] != ',')

{

rd[j] = line[k];

j++;

k++;

}

rd[j] = '\0';

k = k + 2;

j = 0; // Read rs

while (line[k] != ',')

{

rs[j] = line[k];

j++;

k++;

}

rs[j] = '\0';

k = k + 2;

j = 0; // Read rt

while (line[k] != ',')

{

rt[j] = line[k];

j++;

k++;

}

rt[j] = '\0';

k = k + 2;

ואת שדה הimmediate

j = 0; // Read immidiate

while (line[k] != ' ')

{

if ((line[k] == '\t') || (line[k] == '#') || (line[k] == '\n'))

break;

imm[j] = line[k];

j++;

k++;

}

imm[j] = '\0';

שמור ברשימה ואז קדם את אינדקס שורת הקוד הנוכחית(בזיכרון) ואת השורה האחרונה.

// Copy Line to MemoryLine structure

head = add\_line(head, option, rd, rs, rt, imm, i);

i++;

if (i > pos1) //Update last line position

pos1 = i;

}

}

בסוף, מייצרים שדה מסוג memory כדי לשמור את השורה האחרונה ואת רשימת שורות הזיכרון.

// create memory structure and return it

Memory\* mem = (Memory\*)malloc(sizeof(Memory));

mem->head = head;

mem->last = pos1;

return(mem); //Return number of lines

}

פונקציית ביניים - החלפת משמות לייבלים למספרי שורות זיכרון:

אחרי שהאסמבלר קורא את הקובץ פעמיים ולפני שהוא כותב את קובץ הפלט. הוא עובר על רשימות שורות הזיכרון והלייבלים ובכל מקום שהוא מוצא לייבל ב-immediate הוא מחליף אותו משם הלייבל למספר שורת הזיכרון שהוא מופנה אליו. הפונ' מקבלת את הרשימות ולא מחזירה כלום

// a label switch function that runs between the second run and the write. changes label names in the memory structure to thier locations taken

// from the label structure

// this function also changes immediate to zero if the register name $zero was recorded in the immediate field

void LableChange(MemoryLine\* head, label\* lb)

{

מחרוזת זמנית לביצוע ההחלפה. תכלול את מספר הלייבל כמחרוזת

char temp[50];

שורת הזיכרון הנוכחית

// the current memory line

MemoryLine \*current = head;

רד עד לסוף הזיכרון

while (current != NULL) {

וחפש אם מה שכתוב בimmediateואם זה לייבל ידוע הכנס את כתובת הזיכרון שהוא מפנה אליו למשתנה loc. יוחזר -1 אם הלייבל לא נמצא או שהimmediate הוא מספר.

// find if there is a label on the immediate and if it exists

int loc = find(lb,current->imm);

// if found

if (loc != -1) {

אם הלייבל כן נמצא. האסמבלר ממיר אותו ממספר למחרוזת ומציב במחרוזת temp

\_itoa(loc, temp, 10); // Changes int to string and puts in temp

והכנס לשדה הimmediate מחרוזת מספרית של מיקום הלייבל

strcpy(current->imm, temp); // Copy label location number to immidiate

}

אם הimmediate הוא $zero(מצב שיכול לקרות בקוד שלא כתוב נכון) אנו משנים אותו ל"0"

if (strcmp(current->imm, "$zero") == 0) // If immidiate is &zero

{

strcpy(current->imm, "0"); // Changes immidiate to "0"

}

לשורה הבאה

current = current->next;

}

}

כתיבת קובץ הפלט - PrintDataToFile

עכשיו שיש לנו רשימה של כל תאי הזיכרון הידועים נכתוב את הכל לקובץ הפלט. הפונקציה שעושה זאת מקבלת את מצביע לזיכרון mem הכולל בתוכו את רשימת תאי הזיכרון, ומצביע לקובץ הפלט memin כאשר השם נבחר מהוראות הפרוייקט שכן הקובץ הוא קובץ הפלט של הסימולטור. במקרה ואין מידע על שורת הזיכרון הנוכחית. הקוד כותב שורת אפסים. כמו כן הקוד מבצע פקודות .word המבוצעות בשלב האסמבלר

//gets memory head and output file indicator. prints the memory into file

void PrintDataToFile(Memory\* mem, FILE \*memin)

{

הגדרת משתנים שבהם נעשה שימוש, i אינדקס שורת הזיכרון ו-num מספר אותו כותבים בפקודת .word

// i - memory index

// num - word for .word

int i = 0, num = 0;

לולאה שנגמרת אם קיבלנו קובץ אסמבלי ריק (רשימת זיכרון ריקה) או כאשר i מגיע לסוף הזיכרון

while (mem->head != NULL&& i <= mem->last)

{

כך את המידע על שורת הקוד הנוכחי מהרשימה. זה מבוצע פעם אחת כי כל מעבר על הרשימה לוקח הרבה זמן ריצה וברצונינו ליצור אסמבלר שמסיים את עבודתו מהר ככל האפשר

// get the current line's data once. this will reduce the code's execution time. allowing it to build apps much more quickly

MemoryLine \*currentLine = getAtPos(mem->head, i);

רשום את האופקוד של השורה בקובץ הפלט כמספר הקסאדצימלי דו תווי. אם השורה היא NULL (לא קיים עליה מידע) תרשום אפסים במטרה שכל השורה תהיה אפסים

// Printing Opcode. if data for the ith row does not exist print a zero

{if (currentLine == NULL)

fprintf(memin, "00");

elseif (strcmp(currentLine->opcode, "add") == 0)

fprintf(memin, "00");

elseif(strcmp(currentLine->opcode, "sub") == 0)

fprintf(memin, "01");

elseif(strcmp(currentLine->opcode, "and") == 0)

fprintf(memin, "02");

elseif(strcmp(currentLine->opcode, "or") == 0)

fprintf(memin, "03");

elseif(strcmp(currentLine->opcode, "sll") == 0)

fprintf(memin, "04");

elseif(strcmp(currentLine->opcode, "sra") == 0)

fprintf(memin, "05");

elseif(strcmp(currentLine->opcode, "srl") == 0)

fprintf(memin, "06");

elseif(strcmp(currentLine->opcode, "beq") == 0)

fprintf(memin, "07");

elseif(strcmp(currentLine->opcode, "bne") == 0)

fprintf(memin, "08");

elseif(strcmp(currentLine->opcode, "blt") == 0)

fprintf(memin, "09");

elseif(strcmp(currentLine->opcode, "bgt") == 0)

fprintf(memin, "0A");

elseif(strcmp(currentLine->opcode, "ble") == 0)

fprintf(memin, "0B");

elseif(strcmp(currentLine->opcode, "bge") == 0)

fprintf(memin, "0C");

elseif(strcmp(currentLine->opcode, "jal") == 0)

fprintf(memin, "0D");

elseif(strcmp(currentLine->opcode, "lw") == 0)

fprintf(memin, "0E");

elseif(strcmp(currentLine->opcode, "sw") == 0)

fprintf(memin, "0F");

elseif (strcmp(currentLine->opcode, "reti") == 0)

fprintf(memin, "10");

elseif (strcmp(currentLine->opcode, "in") == 0)

fprintf(memin, "11");

elseif (strcmp(currentLine->opcode, "out") == 0)

fprintf(memin, "12");

elseif (strcmp(currentLine->opcode, "halt") == 0)

fprintf(memin, "13");

במקרה שהתברר שהאופקוד הוא NONO. אנו בודקים אם כל השדות שווים לNONO ובמקרה שכן הבנו שאנחנו בפקודת .word

// if there is no opcode. this block of code is used to get the word for the .word command

elseif(strcmp(currentLine->opcode, "NONO") == 0) {

if (strcmp(currentLine->rd, "NONO") == 0) {

if (strcmp(currentLine->rs, "NONO") == 0) {

if (strcmp(currentLine->rt, "NONO") == 0) {

הופכים את המילה data למספר. אם היא הקסאדצמלית ממירים אותה לבינארית

if ((currentLine->imm[0] == '0') && (currentLine->imm[1] == 'x')) //Check if immidiate in hex

{

num = strtol(currentLine->imm, NULL, 16);

}

else//Imiddiate is decimal

{

num = atoi(currentLine->imm);

}

ואז כותבים את המספר בהקסאדצימלית

//Print immidiate in hex

fprintf(memin, "%08X", num);

}

}

}

}

אם אין כלום – רשום שני אפסים

else// if there is nothing print a zero

fprintf(memin, "00"); }

חלק הקוד הבא ממיר את האוג' rd לתו הקסאדצימלי(4 ביטים) ורושם אותו בזיכרון. במקרה שהוא לא ידוע או שאין מידע על השורה נרשם אפס

// Printing Rd

{if (currentLine == NULL)

fprintf(memin, "0");

elseif (strcmp(currentLine->rd, "$zero") == 0)

fprintf(memin, "0");

elseif(strcmp(currentLine->rd, "$imm") == 0)

fprintf(memin, "1");

elseif(strcmp(currentLine->rd, "$v0") == 0)

fprintf(memin, "2");

elseif(strcmp(currentLine->rd, "$a0") == 0)

fprintf(memin, "3");

elseif(strcmp(currentLine->rd, "$a1") == 0)

fprintf(memin, "4");

elseif(strcmp(currentLine->rd, "$t0") == 0)

fprintf(memin, "5");

elseif(strcmp(currentLine->rd, "$t1") == 0)

fprintf(memin, "6");

elseif(strcmp(currentLine->rd, "$t2") == 0)

fprintf(memin, "7");

elseif(strcmp(currentLine->rd, "$t3") == 0)

fprintf(memin, "8");

elseif(strcmp(currentLine->rd, "$s0") == 0)

fprintf(memin, "9");

elseif(strcmp(currentLine->rd, "$s1") == 0)

fprintf(memin, "A");

elseif(strcmp(currentLine->rd, "$s2") == 0)

fprintf(memin, "B");

elseif(strcmp(currentLine->rd, "$gp") == 0)

fprintf(memin, "C");

elseif(strcmp(currentLine->rd, "$sp") == 0)

fprintf(memin, "D");

elseif(strcmp(currentLine->rd, "$fp") == 0)

fprintf(memin, "E");

elseif(strcmp(currentLine->rd, "$ra") == 0)

fprintf(memin, "F");

elseif(strcmp(currentLine->rd, "NONO") == 0)

num = 0;

else

fprintf(memin, "0"); }

אותו דבר עבור האוגר Rs

// Printing Rs

{if (currentLine == NULL)

fprintf(memin, "0");

elseif (strcmp(currentLine->rs, "$zero") == 0)

fprintf(memin, "0");

elseif(strcmp(currentLine->rs, "$imm") == 0)

fprintf(memin, "1");

elseif(strcmp(currentLine->rs, "$v0") == 0)

fprintf(memin, "2");

elseif(strcmp(currentLine->rs, "$a0") == 0)

fprintf(memin, "3");

elseif(strcmp(currentLine->rs, "$a1") == 0)

fprintf(memin, "4");

elseif(strcmp(currentLine->rs, "$t0") == 0)

fprintf(memin, "5");

elseif(strcmp(currentLine->rs, "$t1") == 0)

fprintf(memin, "6");

elseif(strcmp(currentLine->rs, "$t2") == 0)

fprintf(memin, "7");

elseif(strcmp(currentLine->rs, "$t3") == 0)

fprintf(memin, "8");

elseif(strcmp(currentLine->rs, "$s0") == 0)

fprintf(memin, "9");

elseif(strcmp(currentLine->rs, "$s1") == 0)

fprintf(memin, "A");

elseif(strcmp(currentLine->rs, "$s2") == 0)

fprintf(memin, "B");

elseif(strcmp(currentLine->rs, "$gp") == 0)

fprintf(memin, "C");

elseif(strcmp(currentLine->rs, "$sp") == 0)

fprintf(memin, "D");

elseif(strcmp(currentLine->rs, "$fp") == 0)

fprintf(memin, "E");

elseif(strcmp(currentLine->rs, "$ra") == 0)

fprintf(memin, "F");

elseif(strcmp(currentLine->rs, "NONO") == 0)

num = 0;

else

fprintf(memin, "0"); }

והאוגר Rt

// Printing Rt

{if (currentLine == NULL)

fprintf(memin, "0");

elseif (strcmp(currentLine->rt, "$zero") == 0)

fprintf(memin, "0");

elseif(strcmp(currentLine->rt, "$imm") == 0)

fprintf(memin, "1");

elseif(strcmp(currentLine->rt, "$v0") == 0)

fprintf(memin, "2");

elseif(strcmp(currentLine->rt, "$a0") == 0)

fprintf(memin, "3");

elseif(strcmp(currentLine->rt, "$a1") == 0)

fprintf(memin, "4");

elseif(strcmp(currentLine->rt, "$t0") == 0)

fprintf(memin, "5");

elseif(strcmp(currentLine->rt, "$t1") == 0)

fprintf(memin, "6");

elseif(strcmp(currentLine->rt, "$t2") == 0)

fprintf(memin, "7");

elseif(strcmp(currentLine->rt, "$t3") == 0)

fprintf(memin, "8");

elseif(strcmp(currentLine->rt, "$s0") == 0)

fprintf(memin, "9");

elseif(strcmp(currentLine->rt, "$s1") == 0)

fprintf(memin, "A");

elseif(strcmp(currentLine->rt, "$s2") == 0)

fprintf(memin, "B");

elseif(strcmp(currentLine->rt, "$gp") == 0)

fprintf(memin, "C");

elseif(strcmp(currentLine->rt, "$sp") == 0)

fprintf(memin, "D");

elseif(strcmp(currentLine->rt, "$fp") == 0)

fprintf(memin, "E");

elseif(strcmp(currentLine->rt, "$ra") == 0)

fprintf(memin, "F");

elseif(strcmp(currentLine->rt, "NONO") == 0)

num = 0;

else

fprintf(memin, "0"); }

כעת, נרשום את ה-immediate. לפני כן אנו עורכים בדיקה האם יש צורך בכך. אם זה .word הimmediate כבר נרשם ולכן מדלגים, ואם השורה היא null(אין מידע עליה) מדלגים

// a "boolean" for wheter to print the immediate

int printnum = 1;

// skip .word lines

if (currentLine != NULL&& strcmp(currentLine->opcode, "NONO") == 0) {

printnum = 0;

}

if (printnum) // now print if satisfied

{

לבסוף רושמים את ה-immediate. כמובן שהוא אפס אם אין מידע על השורה

// on a null line. the immediate is zero

if (currentLine == NULL) {

num = 0;

}

אחרת ממירים אותו למספר. יש המרה גם ל-immediate הקסאדצימלי וגם ל-immediate בינארי

elseif ((currentLine->imm[0] == '0') && (currentLine->imm[1] == 'x')) //Check if immidiate in hex

{

num = strtol(currentLine->imm, NULL, 16);

}

else//Imiddiate is decimal

{

num = atoi(currentLine->imm);

}

וכותבים אותו בהקסאדצימלי בקובץ הפלט, ה-& 0xfff אמור לקצר מספרים שליליים לאורך מתאים

//Print immidiate in hex. the & 0xfff is supposed to shorte negative numbers to 3 hexadecimal digits or 12 bits

fprintf(memin, "%03X", num & 0xfff);

}

כתוב "שורה חדשה" כדי לעבור לשורה הבאה בזיכרון

//Print \n except the last line

fprintf(memin, "\n");

// go to next line

עדכן את אינדקס השורה

i++;

}

}

הפונקציה הראשית –main:

פונקציה קצרה זו היא הפונקציה הראשית אותה מריץ האסמבלר בעת קריאתו, בין אם מהcmd או מהvisual studio. היא מקבלת מספר argc שהינו קבוע 3 ומערך של שלוש מחרוזות argv ובו המחרוזת הראשונה היא שם התכנה, השנייה היא שם קובץ הקלט והשלישית היא שם קובץ הפלט

// the main takes two arguments, the input file and the output file. indexes start with 1 because argv[0] is the program itself

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

פתח את קובץ הקלט פעם ראשונה

// open the input file. doing so in the main function will allow us to have infinite length file names

// why i call it "asembl"? because of what it is

FILE \*asembl = fopen(argv[1], "r");

// leave if null file is supplied

במקרה שלא נפתח, יוצאים

if (asembl == NULL) {

exit(1);

}

// the first iteration, locate the labels and write thier locations to the linked list

יוצרים את רשימת הלייבלים(מעבר ראשון)

label\* labels = createLabelList(asembl);

// close the file from the first iteration

סוגרים ופותחים מחדש את קובץ הקלט כדי לחזור להתחלה.

fclose(asembl);

// and reopen it for the second

asembl = fopen(argv[1], "r");

// another null check in case something happend

if (asembl == NULL) {

exit(1);

}

עוברים על הקובץ פעם שניה ומחזירים את מבנה הזיכרון

// start the second iteration

Memory \*memory = SecondRun(asembl);

מחליפים את הלייבלים בכתובות בזיכרון וסוגרים את קובץ הקלט סופית

fclose(asembl);

LableChange(memory->head, labels); // Change labels from words to numbers

פה יוצרים את קובץ הפלט(ודורסים אותו אם הוא קיים) ואם לא מצליחים, יוצאים מהקוד

// Write Data to file

FILE\* memin = fopen(argv[2], "w");

if (memin == NULL)

exit(1);

לבסוף כותבים את קובץ הפלט

PrintDataToFile(memory, memin);

סוגרים אותו כדי שתוכנות אחרות יוכלו להיכנס אליו(כגון הסימולטור)

fclose(memin);

// End of file writing

ולבסוף, מנקים את כל הזיכרון של המבנים(סוג של garbage collection ידני)

// free the memory taken by the label list and memory structure

destroy(labels);

destroy\_mem(memory);

}

עם כל הקודים האלה, האסמבלר מבצע את עבודתו וממיר קבצי אסמבלי לקובצי שפת מכונה הקסאדצימלים עבור

**הסימולטור –simulator.exe**