מבנה המחשב – פרוייקט ISA, דוקומנטציה

איתי מרקמן – 206900193

עידו גפן - 318796075

בן דב בלוך - 314829219

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	$R[rd] = R[rs] + R[rt]$
01	Sub	$R[rd] = R[rs] - R[rt]$
02	And	$R[rd] = R[rs] \& R[rt]$
03	or	$R[rd] = R[rs] \| R[rt]$
04	Sll	$R[rd] = R[rs] \ll R[rt]$
05	Sra	הזזה אריתמטית עם הארכת סימן $R[rd] = R[rs] \gg R[rt]$
06	Srl	$R[rd] = R[rs] \gg R[rt]$ הזזה לוגית
07	beq	$if\ (R[rs] == R[rt]) pc = R[rd]$
08	bne	$if\ (R[rs] != R[rt]) pc = R[rd]$
09	blt	$if\ (R[rs] \leq R[rt]) pc = R[rd]$
0A	bgt	$if (R[rs] > R[rt])\ pc = R[rd]$
0B	Ble	$if (R[rs] \leq R[rt])\ pc = R[rd]$
0C	bge	$if (R[rs] \geq R[rt])\ pc = R[rd]$
0D	jal	$R[15] = pc + 1, pc = R[rd][0:11]$
0E	lw	$R[rd] = MEM\big[R[rs] + R[rt]\big]$
0F	sw	$MEM\big[R[rs] + R[rt]\big] = R[rd]$
11	reti	$pc = IORegister[7]$
12	in	$R[rd] = IORegister\big[R[rs] + R[rt]\big]$

13	out	$IORegister[R[rs]+R[rt]] = R[rd]$
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 ומemi 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	$\$zero$	תמיד מכיל אפס
1	$\$imm$	הפניה לשדה ה-immediate של הפקודה
2	$\$v0$	ערך מוחזר מפונקציה
3	$\$a0$	ארגומנט של פונקציה
4	$\$a1$	ארגומנט של פונקציה
5	$\$t0$	משתנה זמני
6	$\$t1$	משתנה זמני
7	$\$t2$	משתנה זמני
8	$\$t3$	משתנה זמני
9	$\$s0$	משתנה זמני נשמר
A	$\$s1$	משתנה זמני נשמר
B	$\$s2$	משתנה זמני נשמר
C	$\$gp$	Global pointer(static data)
D	$\$sp$	מצביע למחסנית
E	$\$fp$	Frame pointer
F	$\$ra$	כתובת חזרה לאחר שימוש בפונקציה

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

<u>האסמבלר –Assembler.c</u>

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

<u>מבנה עזר –label:</u>

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 זכרון, 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

<u>מבנה עזר - שורת זיכרון MemoryLine:</u>

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)
       return NULL;
// 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
Return head;
}
```

פונקציה שמוסיפה שורת זיכרון נוספת לרשימה. היא משתמשת במצביע tail על מנת להוסיף את השורה בסוף הרשימה. דבר שחשבנו שיקל 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
        MemoryLine* 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

<u>מבנה עזר - זיכרון –Memory:</u>

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 שתי פונקציות. שכן השימוש בפקודות שרלוונטיות אליו מועט. פונקציה היוצרת זכרון ומחזירה אותו ופונקציה 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);
}
```

זוהי הפונקציה המשמידה את הזכרון על ידי שחרור הרשימה

```
//create memory structure
Memory* create_mem(MemoryLine* head, int pos1)
{
        Memory* mem = (Memory*)malloc(sizeof(Memory));
        mem->head = head;
        mem->last = pos1;
        return(mem); //Return number of lines
}
```

וזוהי הפונקציה היוצרת זיכרון דינאמי ומחזירה אותו

<u>איטרציות על הקבצים:</u>

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

<u>איטרציה ראשונה – יצירת רשימת הלייבלים:</u>

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 מסבירה על תפקידו של כל משתנה.

```
// rowIndex-the code row's index. where the PC will go after reading the label
// k is the char index for label name read, j is the index in the label name string we
are building, option is determining if it's a label only line or a label + command line
// counter will be the line number in the new hexadecimal code. it will go up when a
line that gets translated is found
int rowIndex = 0, k, j, option, counter = 0;
// line the current line being read, tav1 is the first char and i used to check for
remarks, // tav - current char when reading label name
// label_line will contain the name of the label once iteration is complete, dots are
used to say "this is a label"
char line[MAX_LINE], tav1 ,tav, lable_line[50], dots[50];
label* head = NULL;  // the label list's head
```

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 קודם הוא עורך בדיקה האם הנקודותיים נמצאות בתוך הערה מעבר לתו # ואז זו אזעקת שווא ועוברים לשורה הבאה

```
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

```
k = 0; j = 0; //Read the label name, first reset indexes

do {
// get current char
tav = line[k];
if (tav != ':') {
         if (tav != '\t' && tav != ' ')    // don't read tabs and spaces
                         {
// 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 לכן נוסיף אותו.

```
// 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 ואחריו פקודה) 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;
}
```

<u>איטרציה שניה – רשימת שורות קובץ הזיכרון:</u>

פונקציות עזר לאיטרציה השנייה:

תפקיד הפונקציה specialworld היא לטפל בפקודות word. כלומר לשמור בזכרון.

הפונקציה שומרת את ערך המיקום בזכרון ואת הערך השמור על ידי מעבר תו אחרי תו.

Head- רשימת מיקומי השורות

Line- השורה אותה אנו קוראים עכשיו

Pose1- אינדקס מיקום סוף השורה

K – התו אותו אנו קוראים

```
MemoryLine *specialword(MemoryLine* head, char line[MAX_LINE], int *pos1, int *k) {
char wordP[15], wordN[15]; // wordP - address, wordN - data
```

קודם בוצעה פעולת דילוג על רווחים.

```
int j = 0; // index for string we copy to
*k = 0; // reset k index
while (line[*k] != ' ') *k=*k+1;// go past all the spaces
*k = *k + 1;
j = 0; //Copy Address. first reset j then copy char by char until the next space
```

בחלק זה נבצע העתקה של מיקום הזכרון אל המערך המייצג את מיקום הזכרון

```
while (line[*k] != ' ') {
        wordP[j] = line[*k];
        j = j + 1; *k = *k + 1;
}
```

באותו אופן בדיוק מבוצעת העתקה של הערך שצריך לשמור למערך המייצג אותו.

```
wordP[j] = '\0'; *k = *k + 1;  // terminate string with null and increment end to next
char
j = 0; //Copy Data. using the same way.
```

```
while (line[*k] != ' ')
{
        if (line[*k] == '\n') break;// but detect an end of line string because after
the data there can be a line end
                wordN[j] = line[*k];
                j = j + 1; *k = *k + 1;
}
wordN[j] = '\0';
```

בחלק זה מתבצעת קביעה האם הערך נתון כמספר דצימלי או הקסאדצימלי וטיפול לכל מקרה.

```
int pos = 0; // pos - current line address, can be hexadecimal or decimal
if (wordP[0] == '0') { //  change Address int. the if block considers an hexadecimal
input
        if (wordP[1] == 'x' || wordP[1] == 'X') pos = strtol(wordP, NULL, 16);
}
else pos = atoi(wordP); // and the else blocks considers a decimal input
```

בחלק זה נבצע את השמירה זכרון עצמו, נתשמש ב NONO. בתור אינדקציה להמשך שמדובר בפעולת word.

```
// 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
        char nono[5] = "NONO"; // a string used to copy nono to required places. fifth
char is null
        strcpy(nono, "NONO"); strcpy(nono, "NONO"); strcpy(nono, "NONO"); strcpy(nono,
"NONO");
        head = add_line(head, nono, nono, nono, nono, wordN, pos);  // save line to
line list. wordN - the immediate value, is used as the data
        if (pos > *pos1) *pos1 = pos;  // update the location of the end of the memory
        return head;
```

לבסוף נחזיר את רשימת השורות העדכנית.

```
}
```

פונקציות readorder קוראת את הפקודה שוב על ידי מעבר תו אחרי תו , ושומרת אותה במערך עזר option.

```
void readorder(char line[MAX_LINE], char *option, int *k){

        char tav; // current read char at index k
        int j = 0; // index of copied char
```

נבצע את הקריאה עד שנזהה כי הגענו לשימוש ברגיסטר

```
        do { // reading opcode should continue till dollar of first register
                tav = line[*k]; // read current
                if (tav != '$') // if it's not dollar
                {
                        if (tav != '\t') // or whitespace
                                if (tav != ' ')
                                {
                                        option[j] = tav; // copy
                                        j = j + 1;
                                }
                        *k = *k + 1;
                }
        } while (tav != '$');
        option[j] = '\0'; // null terminate the opcode
```

וכמובן בסוף נוסיף תו סיום כמו כל מחרוזת בשפת סי

הפונקציה הבאה תקדם אותנו עד לרגיסטר, פונקציה פשוטה המבצעת פעולה שחוזרת על עצמה.

```
}
void readdollar(char line[MAX_LINE],int *k){
        while (line[*k] != '$') *k = *k + 1; // simply make your way to the dollar then
stop
}
```

פונקציה זו עוברת על הרגיסטר ושומרת את ערכו במערך. היא מקבלת את סוגו ובכך יודעת מה הוא, בסוף מחזירה את המערך.

```
// reads register value
// line - current line being read
// rdst - register name. named so it can be used for rd, rs or rt
```

```
void readrdst(char line[MAX_LINE], char *rdst, int *k){
        int j = 0; // Read rd
        while (line[*k] != ',')
        {
                if (line[*k] != ' ' && line[*k] != '\t') { // read if not a whitespace
                        rdst[j] = line[*k];
                        j=j+1;
                }
                *k=*k+1;
        }
        rdst[j] = '\0';// null terminate
        return rdst; // return the string
}
```

הפונקציה הבאה עוברת על ערך הimm שומרת ומחזירה אותו במערך

```
void readimmd(char line[MAX_LINE], char *imm, int *k){
        // go to immediate
        while ((line[*k] == ' ') || (line[*k] == '\t') || (line[*k] == ',')) {
                *k = *k + 1;
        }
        int j = 0; // index of char being copied in immediate string
        while (line[*k] != ' ')
        {
                if (line[*k] != ' ' && line[*k] != '\t') {
                        if ((line[*k] == '\t') || (line[*k] == '#') || (line[*k] ==
'\n')) break;
                        imm[j] = line[*k];
                        j=j+1;
                }
                *k=*k+1;
        }
        imm[j] = '\0';
        return imm;
}
```

מבצעת את קריאת השורה לזכרון, היא משתמשת בפונקציות הקודומות ושומרת את התוצאות ברשימת readLineהפונקציה-
) ומחזירה אותה לאחר מכן.Headשורות הזכרון(

```
// reads line of memory and adds to memory line list "head
// line - what we read
// pos1 - number of last line of memin
// i - current instruction line index. might be more than pos1. is processed as we go
so that's why a pointer
// head - the memory line list we add to
// k - index of char being read. is processed as we go so that's why a pointer
MemoryLine *readLine(char *line, int *pos1, int *i, MemoryLine *head, int *k) {
        char option[6], rd[6], rs[6], rt[6], imm[50]; // the line's properties
        readorder(line, option, k);            // read the opcode
        readrdst(line, rd, k);                     // Read rd
        readdollar(line, k);                       // wait for dollar sign
        readrdst(line, rs, k);                     // Read rs
        readdollar(line, k);                      // wait for dollar sign
        readrdst(line, rt, k);                     // Read rt
        readimmd(line, imm, k);                  //handle immediate
        head = add_line(head, option, rd, rs, rt, imm, *i);   *i = *i + 1;
        if (*i > *pos1) *pos1 = *i;  //Update last line position
        return head;
}
```

כעת הגענו אל האיטרציה השנייה ממש, איטרציה זו גם כן עוברת על קוד האסמבלי, אך היא בודקת עבור כל שורה בקובץ איך
הינו מצביע לקובץ הקלטfileהיא משפיעה על קובץ הזיכרון. ומכניסה את המידע לרשימת שורות הזיכרון. המצביע

```
Memory* SecondRun(FILE* file) {
    // k - the index of the current char being read, i - the current position in the
file
        // pos1 - the last line of the memory file
int k = 0, i = 0, pos1 = 0;
        // char - line will house the current line. tav1 will save the first character
of the line and option, rd, rs, rt and are the command's values
        // dots - used to detect labels. because something might be past them
char line[MAX_LINE], tav1, * dots = ":";
```

```
MemoryLine* head = NULL;  // the Memory list's head. it will contain info about each
memory line in the end
while ((fgets(line, MAX_LINE, file)) != NULL) { // the loop reads the file line by
line. and upon reaching null it stops as that's where the file ends
        if (strcmp(line, "\n") == 0) continue;  // in case of a Blank line, go
        tav1 = line[0];          // get first line
        if (tav1 == '#') continue;  // in case of a Remark line, go
```

עד חלק זה מתבצעת הגדרת משתנים ודילוג על שורות ריקות או בעלות הערות בלבד

```
                char wo[6] = ".word"; // a string for comparison
                int isword = 0; // booleand for .world detection
```

במקרה וזוהי אכן פקודה מיוחדת השורות הבאות יטפלו בה word.ערכים הבודקים האם זוהי פקודה מיוחדת של

```
                if (strstr(line, wo) != NULL) { // in case of the special .word order
                        head = specialword(head, line, &pos1, &k);
                        isword = 1;
                }
```

עבור פקודות רגילות עם לייבל נבדוק האם הנקודותיים הם הערה ולא לייבל, במקרה וזוהי אכן שורות לייבל נבדוק גם האם זוהי רק שורת לייבל או גם הוראה

```
                else if (strstr(line, dots) != NULL){     //in case of regular order
and label
                        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
                                        goto mark; // start reading
                                }
                        } // the following code section will work if the there is mark #
and it appears after dots or if there isn't mark #
                        k = 0;
                        while (line[k] != ':') k++;
                        k++;
                        if (line[k] == '\n') continue;
                        else
                                while ((line[k] == ' ') || (line[k] == '\t')) k++;
                        if (line[k] == '\n') continue;
                        if (line[k] == '#')    continue;
                        if (i > pos1) pos1 = i;  //Update last line position
                }
                else// Order line only
                {
                        k = 0;
                        while ((line[k] == ' ') || (line[k] == '\t')) k++;  // roll to
end of spaces
                        if (line[k] == '#')    continue;
                        if (line[k] == '\n') continue;
                }
```

חלק זה מבצע את העתקת השורה במקרה וזוהי לא word command.

```
                if (!isword) { // copy line in all not .world scenarios
                mark:
                        head = readLine(line, &pos1, &i, head, &k);
                }
        }
```

ולבסוף ניצור את הזכרון של הקובץ.

```
        return create_mem(head, pos1);  // create memory structure and return it to
main function
}
```

<u>הפונקציות הבאות עוזרות להדפיס לקובץ הסופי את הדרוש:</u>

הפונקציה pringrdrsrt עוזרת לנו להדפיס את שמות הרגיסטרים, בפועל ירשם מספר המייצג את שם הרגיסטר.

```
// rdst - register name to print
// memin - current file pointer
// num - will be flipped to zero if it's a .world
void printrdrsrt(char *rdst, FILE *memin, int *num)
{ // basically a big if block that checks the name of the register and converts it to a
number
        if (strcmp(rdst, "$zero") == 0)
                fprintf(memin, "0");
```

```
        else if (strcmp(rdst, "$imm") == 0)
                fprintf(memin, "1");
        else if (strcmp(rdst, "$v0") == 0)
                fprintf(memin, "2");
        else if (strcmp(rdst, "$a0") == 0)
                fprintf(memin, "3");
        else if (strcmp(rdst, "$a1") == 0)
                fprintf(memin, "4");
        else if (strcmp(rdst, "$t0") == 0)
                fprintf(memin, "5");
        else if (strcmp(rdst, "$t1") == 0)
                fprintf(memin, "6");
        else if (strcmp(rdst, "$t2") == 0)
                fprintf(memin, "7");
        else if (strcmp(rdst, "$t3") == 0)
                fprintf(memin, "8");
        else if (strcmp(rdst, "$s0") == 0)
                fprintf(memin, "9");
        else if (strcmp(rdst, "$s1") == 0)
                fprintf(memin, "A");
        else if (strcmp(rdst, "$s2") == 0)
                fprintf(memin, "B");
        else if (strcmp(rdst, "$gp") == 0)
                fprintf(memin, "C");
        else if (strcmp(rdst, "$sp") == 0)
                fprintf(memin, "D");
        else if (strcmp(rdst, "$fp") == 0)
                fprintf(memin, "E");
        else if (strcmp(rdst, "$ra") == 0)
                fprintf(memin, "F");
        else if (strcmp(rdst, "NONO") == 0)
                *num = 0;
        else
                fprintf(memin, "0");
}
```

הפונקציה הבאה מדפיסה את הפקודה (שוב על ידי מספר המייצג את סוג הפעולה) ולבסוף מחזירה ערך בוליאני שאחראי להודיע לנו האם בוצעה הדפסה.

```
// opc - opcode string
// memin - output file pointer
int printopcode(char *opc, FILE *memin)
{
        if (strcmp(opc, "add") == 0) {
                fprintf(memin, "00"); return 1;}
        else if (strcmp(opc, "sub") == 0) {
                fprintf(memin, "01"); return 1;}
        else if (strcmp(opc, "and") == 0) {
                fprintf(memin, "02"); return 1;}
        else if (strcmp(opc, "or") == 0) {
                fprintf(memin, "03"); return 1;}
        else if (strcmp(opc, "sll") == 0) {
                fprintf(memin, "04"); return 1;}
        else if (strcmp(opc, "sra") == 0) {
                fprintf(memin, "05"); return 1;}
        else if (strcmp(opc, "srl") == 0) {
                fprintf(memin, "06"); return 1;}
        else if (strcmp(opc, "beq") == 0) {
                fprintf(memin, "07"); return 1;}
        else if (strcmp(opc, "bne") == 0) {
                fprintf(memin, "08"); return 1;}
        else if (strcmp(opc, "blt") == 0) {
                fprintf(memin, "09"); return 1;}
        else if (strcmp(opc, "bgt") == 0) {
                fprintf(memin, "0A"); return 1;}
        else if (strcmp(opc, "ble") == 0) {
                fprintf(memin, "0B"); return 1;}
        else if (strcmp(opc, "bge") == 0) {
                fprintf(memin, "0C"); return 1;}
```

```
        else if (strcmp(opc, "jal") == 0) {
                fprintf(memin, "0D"); return 1;}
        else if (strcmp(opc, "lw") == 0) {
                fprintf(memin, "0E"); return 1;}
        else if (strcmp(opc, "sw") == 0) {
                fprintf(memin, "0F"); return 1;}
        else if (strcmp(opc, "reti") == 0) {
                fprintf(memin, "10"); return 1;}
        else if (strcmp(opc, "in") == 0) {
                fprintf(memin, "11"); return 1;}
        else if (strcmp(opc, "out") == 0) {
                fprintf(memin, "12"); return 1;}
        else if (strcmp(opc, "halt") == 0) {
                fprintf(memin, "13"); return 1;}
        else // on .word
                return 0;
}
```

הפונקציה PrintDataToFile מבצעת את ההדפסה לקובץ. היא משתמשת בהרבה מפונקציות העזר על מנת לבצע את ההדפסה. היא עוברת שורה שורה עד לאחרונה בקובץ. בכל איטרציה היא מדפיסה את הפקודה, אם ואין פקודה היא תדפיס 00 ואילו אם לא התבצעה כתיבה(מצביע על word command.) תטפל גם במקרה זה. לאחר מכן היא עוברת להדפיס את הרגיסטרים, עבור כל אחד מהם היא בודקת האם ישנו אם לא היא תדפיס 0 ואם כן תדפיס את הרגיסטר. לבסוף היא מטפלת במקרה של פקודה מיוחדת של word.

```
//gets memory head and output file indicator. prints the memory into file
void PrintDataToFile(Memory* mem, FILE *memin)
{
        // i - memory index, num - word for .word
        int i = 0, num = 0; int flag=0;
        while (mem->head != NULL && i <= mem->last)   {
                MemoryLine *currentLine = getAtPos(mem->head, i); // get the current
line's data once. this will reduce the code's execution time. allowing it to build apps
much more quickly
                // Printing Opcode. if data for the ith row does not exist print a zero
                if (currentLine == NULL) fprintf(memin, "00");// if no opcode print 2
zeros
                else flag=printopcode(currentLine->opcode, memin); // print the opcode
and return if it was printed
                if (!flag && currentLine != NULL) {// if there is no opcode. this block
of code is used to get the word for the .word command
                        if ((strcmp(currentLine->opcode, "NONO") == 0)) {
                                if ((currentLine->imm[0] == '0') && ((currentLine-
>imm[1] == 'x') || (currentLine->imm[1] == 'X')))
                                        num = strtol(currentLine->imm, NULL, 16);
                                else  //Imiddiate is decimal
                                        num = atoi(currentLine->imm);
                                fprintf(memin, "%08X", num);  //Print immidiate in hex
                        }
                }
                else if (!flag) // if there is nothing print a zero
                        fprintf(memin, "00");
                if (currentLine == NULL) fprintf(memin, "0"); // Printing Rd
                else printrdrsrt(currentLine->rd, memin,&num);
                if (currentLine == NULL) fprintf(memin, "0");                //
Printing Rs
                else printrdrsrt(currentLine->rs, memin,&num);
                if (currentLine == NULL) fprintf(memin, "0");                //
Printing Rt
                else printrdrsrt(currentLine->rt, memin,&num);
                // a check wheter to print the immediate and skip .word lines
                if (currentLine == NULL) fprintf(memin, "%03X", 0 & 0xfff); // on a
null line. print zero to immediate
                else if(strcmp(currentLine->opcode, "NONO") != 0)  // now print if
satisfied
                {
                        if ((currentLine->imm[0] == '0') && ((currentLine->imm[1] ==
'x') || (currentLine->imm[1] == 'X'))) num = strtol(currentLine->imm, NULL, 16);
//Check if immidiate in hex
                        else num = atoi(currentLine->imm);
```

```
                        fprintf(memin, "%03X", num & 0xfff);  //Print immidiate in hex.
the & 0xfff is supposed to shorte negative numbers to 3 hexadecimal digits or 12 bits
                }
                if (i != mem->last) fprintf(memin, "\n");  //Print \n except the last
line
                i++;  // go to next line
        }
}
```

הפונקציה LableChange דואגת לבצע החלפה בין שם הלייבל למיקמו המספרי בקובץ. בנוסף היא מבצעת החלפה בין שם רגיסטר האפס ($zero) ל0.

```
// a label switch function that runs between the second run and the write. changes
label names in the memory structure to their locations taken
// from the label structure
// this function also changes immediate to zero if the register name $zero was recorded
in the immediate field
void LableChange(MemoryLine* head, label* lb)
{
        char temp[50];
        // the current memory line
        MemoryLine *current = head;
        while (current != NULL) {
                // find if there is a label on the immediate and if it exists
                int loc = find(lb,current->imm);
                // if found
                if (loc != -1) {
                        _itoa(loc, temp, 10); // Changes int to string and puts in temp
                        strcpy(current->imm, temp); // Copy label location number to
immidiate
                }
                if (strcmp(current->imm, "$zero") == 0) // If immidiate is &zero
                {
                        strcpy(current->imm, "0"); // Changes immidiate to "0"
                }
                current = current->next;
        }
}
```

ולבסוף יש לנו את החלק המרכזי המבצע את הפעולה כולה על ידי קריאה לפונקציות העזר:
הוא פותח את הקובץ. אחר כך הוא מבצע את הקריאה הראשונה, כלומר יצירת רשימת הלייבלים. לאחר מכן הוא מבצע את האיטרציה השנייה שהיא בניית הזכרון והחלפת שמות הלייבלים לכתובות מתאימות בזכרון, וכמובן סוגרת את הקובץ לאחר מכן. לבסוף התוכנית פותחת את הקובץ אליו היא כותבת, מבצעת את הכתיבה וסוגרת את הקובץ. בפעולות האחרונות מתבצעות שחרור הזכרון הדינאמי שיצרנו.

```
// part 3 - the main function

// 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[]) {
    // 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
    if (asembl == NULL) {
        exit(1);
    }
    // the first iteration, locate the labels and write thier locations to the linked
list
    label* labels = createLabelList(asembl);
    // close the file from the first iteration
    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);
    }
    // start the second iteration
    Memory *memory = SecondRun(asembl);
```

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

        // Write Data to file
        FILE* memin = fopen(argv[2], "w");
        if (memin == NULL)
                exit(1);
        PrintDataToFile(memory, memin);
        fclose(memin);
        // End of file writing

    // free the memory taken by the label list and memory structure
    destroy(labels);
    destroy_mem(memory);

      fclose(memin);
      // End of file writing
// free the memory taken by the label list and memory structure
    destroy(labels);
    destroy_mem(memory);
```

simulator.c-הסימולטור

תפקיד הסימולטור הוא כמו שמו, לבצע סימולציה של הריצה. הוא עוקב אחר ההוראות, אחר מיקום התוכנית, מבצע את ההדפסות הדרושות לקבצים הסופיים ובכך מסמלץ את הקוד.

הסימולטור אוסף שלב אחרי שלב את הערכים אותם הוא צריך על מנת לבצע את הסימולציה ואז מבצע אותה.

פונקציות חשובות:

הפרוצדורה paraseirq2 יוצרת מערך דינאמי המכיל את זמני הפעלת הפקודה החיצונית irq2 היא מקבלת את הקובץ המכיל זמנים אלו ומחזירה את הזמנים כמערך דינאמי של מספרים, מערך זה ישוחרר בהמשך כאשר הסימולטור יבצע halt.

```
// gets irq2
int* parseirq2(FILE* irq2in) {
        // leave is irq2 file didn't open
        if (irq2in == NULL) {
                exit(1);
        }
        // the current row
        char current[MAX_LINE];
        // count the number of ints needed for the required memory
        irq2count = 0;
        while (!feof(irq2in)) {
                fgets(current, MAX_LINE, irq2in);
                irq2count++;
        }
        // return to the start for the actual reading
        fseek(irq2in, 0, SEEK_SET);
        // initialize array as dynamic memory. which will be released when the
simulator halts
        int* irq2 = (int*)malloc(sizeof(int) * irq2count);
        // create index
        int index = 0;
        // fseek resets end of file
        while (!feof(irq2in)) {
                fgets(current, MAX_LINE, irq2in);
                // add to array
                irq2[index] = atoi(current);
                index = index + 1;
        }
        return irq2;
}
```

הפונקציה updateTimer מקבלת מצביע אל מערך רגיסטרי הקלט פלט ומקדמת את השעון לפי הצורך.

```
// basically updates the timer and trips IRQ0. a pointer to the input/output register
array
// is the only input
void updateTimer(int* ioRege) {
        // reset irqstatus0 if timer did not tick
        ioRege[3] = 0;
        // check timer enable before updating
        if (ioRege[11] != 0 && ioRege[12] < ioRege[13]) {
                ioRege[12]++;
                // now we check if the timer event happend
                if (ioRege[12] == ioRege[13]) {
                        ioRege[3] = 1;
                        ioRege[12] = 0;
                }
        }
}
```

הפונקציה Print_To_Trace מדפיסה אל הקובץ trace את ערכי הpc השורה ואת הרגיסטרים וכו בפורמט הנכון, היא מקבלת ערכים אלו כמובן. .

```
// function prints to the trace file. it gets the pc. memory line
// and register array and prints a row on the register values
Print_To_Trace(FILE* trace, int pc, char* line, int Reg_Array[]) {
        // hexval - hexadcimal value of current row
        char hexval[9], instruction[8], * inst = instruction;
        // write hexadecimal PC
        sprintf(hexval, "%08X", pc);
        // write hexadecimal pc in trace
        fputs(hexval, trace);
        // write a space to the file
        putc(' ', trace);
        // get instruction directly as was written in memin
        inst = strtok(line, "\n");
        // and write to file
        fputs(inst, trace);
        // space advance on trace file
        putc(' ', trace);
        // now write each register's value into the trace file
        for (int i = 0; i <= 15; i++) {
                sprintf(hexval, "%08x", Reg_Array[i]);
                fputs(hexval, trace);
                if (i != 15) {
                        putc(' ', trace);
                }
        }
        // finish current row on trace file
        putc('\n', trace);
}
```

הפונקציה הבאה מעדכנת את הקבצים leds and display היא מקבלת את מחזור השעון, מצביע אל רגיסטרי הקלט פלט,ערך מספרי המייצג לאיזה קובץ להדפיס ואת קבצי היציאה אותם היא מעדכנת.

```
// updates leds and display files. gets clock cycle, updated HW register, the HW
register array and the leds and display files
void updateLD(int cycle, int regNum, int* ioRege, FILE* leds, FILE* display) {
        char toWrite[100] = "";
        // start with cycle
        _itoa(cycle, toWrite, 10);
        // cibvert register value to hexa
        char regVal[MAX_LINE];
        sprintf(regVal, "%08x", ioRege[regNum]);
        // add the spacebar
        strcat(toWrite, " ");
        strcat(toWrite, regVal);
        // add the new line
        strcat(toWrite, "\n");
```

```
        // check if to use leds or display file and use the pointer to
        // point the correct file
        FILE* fileToUse;
        if (regNum == 9) {
                fileToUse = leds;
        }
        else {
                fileToUse = display;
        }
        // write to file
        fprintf(fileToUse, toWrite);
}
```

הפרוצדורה updatehwRegTrace מבצעת עדכון לקובץ המכיל את ערכי רגיסטרי החומרה(קלט פלט), לפני העדכון היא משנה כמובן לכתיב הקסאדצימלי.

```
// updates the hwRegTrace file
// gets current cycle, action(0 write 1 read), register being used and HW register
array pointer, as well as a pointer to the file
void updatehwRegTrace(int cycle, int action, int reg, int* ioRege, FILE* hwregTrace) {
        // big string for current line, slowly strings will be added to it then it will
be printed
        // to the file after the null wrap
        char toWrite[2000] = "";
        // start with cycle
        _itoa(cycle, toWrite, 10);
        // 1 - read, 0 - write
        if (action == 1) {
                strcat(toWrite, " READ");
        }
        else {
                strcat(toWrite, " WRITE");
        }
        // the names of the IO registers
        char names[18][50] = { " irq0enable "," irq1enable "," irq2enable "," 
irq0status "," irq1status "," irq2status "," irqhandler "," irqreturn "," clks "," leds 
",
                " display "," timerEnable "," timerCurrent "," timerMax "," diskcmd "," 
disksector "," diskbuffer "," diskstatus " };
        // write the register's name
        strcat(toWrite, names[reg]);
        // now convert the register value. the bitmask is an extention to 8 bits
        // string will hold register value as string
        char regVal[MAX_LINE];
        sprintf(regVal, "%08x", ioRege[reg]);
        strcat(toWrite, regVal);
        // add next line
        strcat(toWrite, "\n");
        // finally, write to file
        fprintf(hwregTrace, toWrite);
}
```

הפונקציה diskOperation מבצעת את הכתיבה או הקריאה, היא מקבלת מצביע אל רגיסטרי הקלט פלט ומצביעים אל המיקום בזכרון ובדיסק

היא בודקת האם הדיסק עסוק ואם הוא איננו עסוק, מבצעת את הפעולה לפי הדרוש ממנה. בנוסף היא דואגת לכך שהדיסק יהיה "עסוק" למשך הזמן הדרוש על ידי איפוס שעון הדיסק

```
// performs disk read and write, gets ioRege - pointer to IO registers, out - pointer
to memory and disk - pointer to disk
void diskOperation(int* ioRege, char out[][9], char disk[][9]) {
        // only do if disk isn't busy
        if (ioRege[17] != 0)
                return;
        // translate sector number to line number in file/array
        int sectorStart = ioRege[15] * 128;
        // Reading loop
```

```
        if (ioRege[14] == 1) {
                // 128 iterations because the sector contains 128 rows
                for (int i = 0; i < 128; i++) {
                        // copy disk sector into memory buffer
                        strcpy(out[i + ioRege[16]], disk[i + sectorStart]);
                }
        }
        else { // writing loop
                for (int i = 0; i < 128; i++) {
                        // copy memory buffer into disk sector
                        strcpy(disk[i + sectorStart], out[i + ioRege[16]]);
                }
        }
        // disk is now busy. so reset the timer
        ioRege[17] = 1;
        diskTimer = 0;
}
```

פרוצדורת העזר moveFP לוקחת את מצביע הקובץ memin ואת ה-PC ומזיזה את המצביע לשורה ה-PC בקובץ

```
// a utility function to move the file pointer to the correct pc
void moveFP(FILE *memin, long pc) {
       char line[MAX_LINE];
       fseek(memin, 0, SEEK_SET); //next five lines made for reading the correct line for next
instruction
       int pc_help = pc;
       while (pc_help > 0) {
              fgets(line, MAX_LINE, memin);
              pc_help -= 1;
       }
                                                                                                }
```

מבנה העזר MemoryLine :

מבנה זה דומה מאוד למבנה שהוסבר כבר בחלק האסמבלי (ראה למעלה) אך כעת הוא לא רשימה אלא פשוט הערכים אותם יש בכל שורה. השדה מכיל את ההוראה, הרגיסטרים rd,rs,rt ערך ה imm, ועותק של השורה כמחרוזת.

```
/ this struct will be used to save the memory lines, similar to one in assembler but
not a list
typedef struct MemoryLine {
        // the opcode of the command
        int opcode;
        // the registers
        int rd;
        int rs;
        int rt;
        // immediate value
        int imm;
        // a copy of the line string
        char line[MAX_LINE];
}MemoryLine;
```

טיפול בפקודות (opcosde)

הטיפול בפקודות בקוד נעשה על ידי פונקציות עזר רבות אשר כל אחת מהן מטפלת בפקודה שונה, צורה זו נעשתה בעיקר לנוחות קריאה ולקיצור אורך פונקציות בקוד. הן כולן פועלות בצורה זהה. מקבלות מצביע לשורה הנוכחית, ערך הpc ומצביע לרגיסטרים. היא מבצעת את הפעולה ומחזירה את ערך הpc העדכני. פקודות הקפיצה (branch) מקבלות בנוסף את הקובץ על מנת לבצע קפיצות בו. ואילו פקודות הכתיבה והקריאה מקבלות גם את הזכרון הנוכחי כמערך. פונקציות אלו קטנות ומאוד פשוטות, ישנו תיאור מעמיק עליהן בקוד עצמו. נציג דוגמאות למבנה כללי:

פעולות אריתמטיות

הפונקציה add מבצעת חיבור בין שני רגיסטרים – היא מקבלת מצביע למיקום שורה בזכרון את ערך ה pc ומצביע למערך הרגיסטרים. היא מחזירה את הpc המעודכן. עבור שאר הפעולות פונקציה דומה פרט לסימן.

```
int add(MemoryLine* current, long pc, int* rege) {
        if (current->opcode == 0) { //ADD instruction
                rege[current->rd] = rege[current->rs] + rege[current->rt];
                pc += 1;
        }
        return pc;
}
```

בפונקציה SRL מתבצעת ההזזה הלוגית בצורה הבאה:

אפס – לא עושים כלום

אחרת – משתמשים במסיכות והזזות מסובכות על מנת שבהזזה ימינה יגיעו אפסים חדשים ולא אחדות עבור מספרים שליליים

```
// this executes "srl" instructions
// current - instruction variables
// pc - current pc
// rege - register array
// returns pc after execution
int srl(MemoryLine* current, long pc, int* rege) {
       if (current->opcode == 6) { //srl instruction, uses a special code
              int size = sizeof(int);
              // shift
              // deal with the zero case
              if (rege[current->rs] == 0)
                     rege[current->rd] = rege[current->rs];
              else /// else use modified >> operator. it uses a bitmask that only lets the original
bits pass to zero any newly created bit
                     rege[current->rd] = (rege[current->rs] >> rege[current->rt]) &
~(((rege[current->rs] >> (size << 3) - 1) << (size << 3) - 1)) >> (rege[current->rt] - 1);
              if (current->rd == 0)
                     rege[current->rd] = 0;
              pc += 1;
       }
       return pc;
}
```

<u>פעולות קפיצה</u>

הפרוצדורה bne מקבלת את אותם הערכים כמו הפעולה הקודמת and, , ובנוסף מקבלת גם את הקובץ על מנת לבצע את ההזזה בקובץ לפי הpc הדרוש. היא מחזירה את ערך הpc החדש, שאר הפעולות מסוג זה עושות דברים דומים פרט לתנאים שונים. כאשר jal עובדת בלי תנאי ומכניסה את הPC האחרון לra

```
int bne(MemoryLine* current, long pc, int* rege, FILE* file) {
        if (current->opcode == 8) {
                if (rege[current->rs] != rege[current->rt]) {
                        pc = rege[current->rd];
                        // move the pointer
                        moveFP(file, pc);
                }
                else
                        pc += 1;
        }
        return pc;
}
```

<u>פעולות שמירה וטעינה</u>

הפרוצדורה lw מבצעת טעינה מזכרון היא מקבלת את אותם הערכים כמו הפרוצדורה add ובנוסף גם מערך ממנו לטעון את הערך, הפונקציה מחזירה את ה pc העדכני כמובן. היא מבצעת המרה ממחרוזת (תווים) למספר ושומרת אותו ברגיסטר rd.

```
int lw(MemoryLine* current, long pc, int* rege, char output[][9]) {
        // stores the memory world we load as an int
        int MEM;
        // and as a string
        char* line2, LINES[MAX_LINE];
        line2 = LINES;
```

```
        if (current->opcode == 14) {   //lw instruction
        // address of word
                int lines = rege[current->rs] + rege[current->rt];
                // get the hexadecimal word
                strcpy(line2, output[lines]);
                // convert to an integer
                MEM = (int)strtol(line2, NULL, 16);
                // put in register
                rege[current->rd] = MEM;
                // zero lock(again)
                if (current->rd == 0)
                        rege[current->rd] = 0;
                pc += 1;
        }
        return pc;
}
```

הפרוצדורה sw לוקחת את אותו מערך כמו lw ומעבירה לו את הערכים מהרג'יסטר rd

```
// this executes "sw" instructions
// current - instruction variables
// pc - current pc
// rege - register array
// output - current memory as an array
// returns pc after execution
int sw(MemoryLine* current, long pc, int* rege, char output[][9]) {
       // and as a string
       char* line2, LINES[MAX_LINE];
       line2 = LINES;
       if (current->opcode == 15) { //sw instruction
              char hexval[9];
              // get line index ffor the store
              int lines = rege[current->rs] + rege[current->rt];
              // convert rd value to hexacdecimal
              sprintf(hexval, "%08X", rege[current->rd]);
              // copy each char in the word to a diffrent bit for the output
              // which will be later written to memout
              output[lines][0] = hexval[0];
              output[lines][1] = hexval[1];
              output[lines][2] = hexval[2];
              output[lines][3] = hexval[3];
              output[lines][4] = hexval[4];
              output[lines][5] = hexval[5];
              output[lines][6] = hexval[6];
              output[lines][7] = hexval[7];
              output[lines][8] = hexval[8];
              pc += 1;
       }
       return pc;
}
```

<u>פעולת קלט פלט</u>

פעולות הקלט פלט מתעסקות ברגיסטרים "חיצוניים" על כן היא מקבל מצביע לרגיסטרים אלו(מספרים חיוביים) וגם לרגיסטרים "הרגילים". בנוסף היא מקבלת קובץ למעקב אחר מחזורי שעון, את מחזור השעון, מצביע למיקום שורה בזכרון ואת ערך הpc. היא מחזירה את ערך הpc המעודכן. במהלך פעולתה היא מבצעת את ההמרה מרגיסטרי החומרה לרגיסטרים הרגילים ומבצעת את הזזת מחזור השעון, כמו כן היא מעדכנת את קובץ המעקב אחרי רג'יסטרי החומרה(updatehwRegTrace)

```
// in operation. gets
// current - memory line struct of instruction variables(registers and immediate and
opcode)
// rege - pointer to register array
// ioRege - pointer to HW register array
// cycle - current clock cycle
// hwregTrace - pointer to hw register array
// pc - current pc
// returns pc after execution
int in(MemoryLine* current, int* rege, unsigned int* ioRege, unsigned int cycle, FILE*
hwregTrace, long pc) {
```

```
        if (current->opcode == 17) {
                // in command. as specified in instructions
                rege[current->rd] = ioRege[rege[current->rs] + rege[current->rt]];
                updatehwRegTrace(cycle, 1, rege[current->rs] + rege[current->rt],
ioRege, hwregTrace);
                pc = pc + 1;
        }
        return pc;
}
```

הפונקציה out אחראית לקריאה לפעולת קלט/פלט על ידי שינוי רג'יסטרי החומרה ומעדכנת את hwregtrace כמו הקודמת. אם השתנו הלדים, הצג או הדיסק קוראים לפונקציה הנכונה על מנת לבצע את הפעולה(לכתוב לקובץ פלט או לבדוק אם אפשר לעשות פעולת דיסק ואז לעשות אותה)

```
// out operation. gets
// current - memory line struct of instruction variables(registers and immediate and opcode)
// rege - pointer to register array
// ioRege - pointer to HW register array
// cycle - current clock cycle
// hwregTrace - pointer to hw register array
// pc - current pc
// output - the current memory as a big array
// disk - the current disk as a big array
// leds - leds file pointer
// display - display file pointer
// returns pc after execution
int out(MemoryLine* current, int* rege, unsigned int* ioRege, unsigned int cycle, FILE* hwregTrace,
long pc, char output[][9], char disk[][9], FILE* leds, FILE* display) {
       if (current->opcode == 18) {
              // update ioRege and hwregtrace file like in in
              ioRege[rege[current->rs] + rege[current->rt]] = rege[current->rd];
              updatehwRegTrace(cycle, 0, rege[current->rs] + rege[current->rt], ioRege, hwregTrace);
              // update leds and display files
              if ((rege[current->rs] + rege[current->rt] == 9) || (rege[current->rs] + rege[current-
>rt] == 10)) {
                     updateLD(cycle, rege[current->rs] + rege[current->rt], ioRege, leds, display);
              }
              // check if disk command register was updated. if so execute the operation
              if ((rege[current->rs] + rege[current->rt] == 14)) {
                     diskOperation(ioRege, output, disk);
              }
              pc = pc + 1;
       }
       return pc;
                                                                                                }
```

הפונקציה הבאה אחראית לבצע את הפעולה לפי ערך הopcode. היא פונקציה פשוטה אשר תבצע רק שורה בודדת מתוך בלוק השורות(ה-if-ים הפנימיים של הפונקציות) ולבסוף תחזיר את מיקום הpc.

```
// performs command as opcode specifies. and returns the pc after it's execution
// current - struct of the line being read
// rege - register array pointer, ioRege - HW register array pointer
// pc - the current pc
// file - memory input file
// output - memory output array
// hwregtrace - hardware register trace file pointer
// cycle - current clock cycle
// leds and display - file pointers to them
// disk - disk content array
int Opcode_Operation(MemoryLine* current, int* rege, unsigned int* ioRege, long pc,
FILE* file, char output[][9], FILE* hwregTrace, unsigned int cycle, FILE* leds, FILE*
display, char disk[][9]) {
        // while MEM is used to convert hexadecimal string to integer in load
        // actually. only one of those functions will execute because an if will take
it out
        pc = add(current, pc, rege);
        pc = sub(current, pc, rege);
```

```
        pc = and (current, pc, rege);
        pc = or (current, pc, rege);
        pc = sll(current, pc, rege);
        pc = sra(current, pc, rege);
        pc = srl(current, pc, rege);
        pc = beq(current, pc, rege, file);
        pc = bne(current, pc, rege, file);
        pc = blt(current, pc, rege, file);
        pc = bgt(current, pc, rege, file);
        pc = ble(current, pc, rege, file);
        pc = bge(current, pc, rege, file);
        pc = jal(current, pc, rege, file);
        pc = lw(current, pc, rege, output);
        pc = sw(current, pc, rege, output);
        pc = reti(current->opcode, ioRege, pc);
        pc = in(current, rege, ioRege, cycle, hwregTrace, pc);
        pc = out(current, rege, ioRege, cycle, hwregTrace, pc, output, disk, leds,
display);

        // prevent changing zero
        if (current->rd == 0)
                rege[current->rd] = 0;

        return pc;
}
```

הפונקציה updateDiskTimer סופרת את שעון הדיסק ומאפסת אותו אחרי 1024 מחזורים כשהדיסק פנוי לפעולה נוספת, בנוסף מבצעת עדכונים לרגסיטרים אם התבצע איפוס של השעון. כל מה שהיא מקבלת זה את המצביע בזכרון לרגיסטרי החומרה. לא מחזירה דבר

```
// updates the "busy disk" timer and resets it on 1024 clock cycles
// gets ioRege - HW register array
void updateDiskTimer(unsigned int* ioRege) {
        // set irq1 status to zero if we can't do anything
        ioRege[4] = 0;
        if (ioRege[17]) {
                diskTimer++;
                // reached 1024 cycles
                if (diskTimer >= 1024) {
                        // reset disk command
                        ioRege[14] = 0;
                        // and disk status
                        ioRege[17] = 0;
                        // and trips irq1status
                        ioRege[4] = 1;
                }
        }
}
```

הפונקציה Print_To_Files מבצעת את ההדפסה לכל קבצי הפלט שנכתבים אחרי הריצה. היא מקבלת מצביעים לכל קבצי הפלט, לרגיסטרים , למספר מחזורי השעון ולזכרון הדיסק

```
// prints all file outputs. gets pointers to all output files. the memout content as an
array. the clock cycle count, the register values at the end and the diskout content as
an array
Print_To_Files(FILE* mem_out, FILE* regout, FILE* trace, FILE* cycles, char
output[][9], unsigned int count, int Reg_Array[], char disk[][9], FILE* diskout) {
```

נשתמש ב i עבור מיקום הזכרון הסופי וב j עבור השורה הנוכחית

```
        // i is where the memory file ends. starts at 65532(very big and moves to end
of file
        // and j is the index of the current row
        int i = MAX_FILE - 2, j = 0;
        //
        char regoutchar[9];
        // move memout end pointer to correct place
        while (strcmp(output[i], "00000000") == 0)
                i -= 1;
        // write memout. with a \n between two subsequent 8 char rows
```

חלק זה מבצע כתיבה לmemout

```
        while (j <= i) {
                fputs(output[j], mem_out);
                putc('\n', mem_out);
                j += 1;
        }
        i = MAX_FILE - 2; // reset i and j to write disk out
        j = 0;
```

ואילו חלק זה אחראי על כתיבה לדיסק

```
        // write disk out same way as memout
        while (strcmp(disk[i], "00000000") == 0)
                i -= 1;
        while (j <= i) {
                fputs(disk[j], diskout);
                putc('\n', diskout);
                j += 1;
        }
        // write the amount of cycles the program ran
        fprintf(cycles, "%u", count);

        // finally. write the register values to regout. not including zero and imm
        for (i = 2; i <= 15; i++) {
                sprintf(regoutchar, "%08X", Reg_Array[i]);
                fputs(regoutchar, regout);
                putc('\n', regout);
```

ולבסוף כתיבת ערכי הרגיסטרים בתצורה הנכונה וכתיבת כמות מחזורי השעון שנעשו

```
        }
}
```

הפרוצדורה checkirq בודקת האם במחזור השעון הנוכחי התרחשה פסיקה, כלומר הטיימר/irq2/דיסק סיים את עבודתו והמעבד צריך לטפל בפסיקה. היא מקדמת את ערך הpc אם הייתה התרחשות כזו ומעבירה את המצביע בקובץ memin כדי לבצע את הפעולה. הפונקציה מקבלת את רגיסטרי החומרה, את ערך הpc העדכני ואת קובץ הmemin .

```
// checks for wheter an irq event has happend. moves pc and trips irqevent "boolean"
// and a pointer to memin
int checkirq(unsigned int* ioRege, int pc, FILE* file) {
        // in case of irq event return current pc
        if (isirqEvent) {
                return pc;
        }
        // the irq boolean as specified
        if ((ioRege[0] && ioRege[3]) || (ioRege[1] && ioRege[4]) || (ioRege[2] &&
ioRege[5])) {
                // return address
                ioRege[7] = pc;
                isirqEvent = 1;
                moveFP(file, ioRege[6]); // move memin file pointer to irq handler
                return ioRege[6];
        }
        // default case - keep current pc
        return pc;
}
```

הפרוצדורה checkirq2 בודקת האם התבצעה קריאה לרגיסטר irq2 במחזור השעון הנוכחי, היא מקבלת מבציע אל רגיסטרי החומרה, את מספר מחזור השעון בו אנו נמצאים ומצביע אל זמני השעון בהם יש קריאה לרגיסטר irq2.

```
// trips irq2 status at the currect clock cycles
// gets pointer to HW registers, clock cycle count and
// the irq2 trigger time array
void checkirq2(unsigned int* ioRege, unsigned int count, int* irq2Times) {
        // reset irq2
        ioRege[5] = 0;
        // loop trough the irq2 to see if event happens
        for (int i = 0; i < irq2count; i++) {
                if (count == irq2Times[i] + 1) {
```

```
                        // trip irq2status
                        ioRege[5] = 1;
                        break;
                }
        }
```

הפונקציה memRead קוראת את קבצי הזכרון והדיסק ומעבירה אותם למערך שבעזרתו נבצע כתיבה אל קובץ הפלט הנחוץ(disk or memout). היא מקבלת את זבכרון הoutput אליו נשמור ומצביע אל זכרון הmemin. היא מחזירה מצביע למערך output. למערכים יש אורך של 65536 מילים. זה מספיק גם לדיסק וגם לזיכרון גם אם בפועל הם אמורים להיות קטנים יותר. על מנת לעכל מערכים כאלה עודכן גודל המחסנית בvisual studio.

```
// reads memory and disk input files(memin) and puts them in array(output)
// the array will be written to memout or diskout eventually
char* memRead(char output[65536][9], FILE* memin) {
```

נגדיר כמה משתנים שיעזרו לנו:

```
        char* out = output;
        // a string for the line being read to output
        char* line, Lines[MAX_LINE];
        // the current memory line
        line = Lines;
        int i = 0;
```

חלק זה מבצע המרה מהזכרון למערך

```
        while (!feof(memin)) { //place input file lines into array
                fgets(line, MAX_LINE, memin);
                // null terminate
                strcpy(output[i], line);
                output[i][8] = '\0';
                i += 1;
        }
        i += 1;
        // place zeros in output until limit
```

ולבסוף נמלא את קובץ הoutput באפסים עד הסוף ונחזיר את מיקום הקריאה מהקובץ ל0.

```
        while (i < MAX_FILE) { // continuing placing lines until limit
                strcpy(output[i - 1], "00000000\0");
                i += 1;
        }
        // reset input file position without closing file
        fseek(memin, 0, SEEK_SET);
        return out;
}
```

הפונקציה Create מעתיקה למבנה הזכרון MemoryLine את ערכי השורה מmemin.

```
// copies to memory line "out" from a line in memin.txt
MemoryLine* Create(FILE *memin, MemoryLine *out) {
        // two strings for opcode and immediate
        char opcode[3], immediate_char[4];
        // get the current memory line
        fgets(out->line, MAX_LINE, memin);
        // get the two char opcode
        strncpy(opcode, out->line, 2);
        opcode[2] = '\0';
        // sting int converter
        out->opcode = (int)strtol(opcode, NULL, 16);
        // rd rs and rt using the single char converter
        out->rd = (int)StrTol(out->line[2]);
        out->rs = (int)StrTol(out->line[3]);
        out->rt = (int)StrTol(out->line[4]);
        // copy immediate value to string
        strncpy(immediate_char, out->line + 5, 3);
        // null terminate
        immediate_char[3] = '\0';
```

```
        // sign extend immediate properly - using arithmetic shifts
        out->imm = (int)strtol(immediate_char, NULL, 16);
        out->imm = out->imm << 20;
        out->imm = out->imm >> 20;
}
```

ולבסוף הגענו לפרוצדורה המרכזית. המain. נעבור שלב שלב בתוכה:

```
// the main function
// argc - number of command arguments(always 12)
// argv - command arguments(names of all the files)
int main(int argc, char* argv[]) {
```

ראשית נבצע הגדרות של מערכים ומצביעים כמו Reg_Array ומצביע אליו. התחלת ערך ה pc וכו.

```
        int Reg_Array[16] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }; //
Reg_Array - array for regular registers
        int* rege = Reg_Array, pc = 0; // Rege - pointer to register array, pc -
current pc, count - current clock cycle, ioRege - pointer to HW register array
        unsigned unsigned int count = 0, IOReg_Array[18] = { 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0 }, *ioRege = IOReg_Array; // count - clock cycle count,
IOReg_array - array for Hardware registers, ioRege - pointer to it
```

נגדיר מצביעים עבור כל הקבצים הרלוונטים.

```
        FILE* memin, * mem_out, * regout, * trace, * hwregtrace, * cycles, * leds, *
display, * diskin, * irq2in, * diskout; // pointers to all the files the simulator is
working with
```

חלק זה פותח ושומר מצביע אל ערכי השעון הרלוונטים(irq2). לאחר מכן הקובץ בונה את המערך לזכרון memout. בנוסף בונה את המערך disk. הערה על גדלי מערכים אלו וההתמודדות איתם למעלה

```
        irq2in = openFile(argv[3], "r"); // open irq2
        int* irq2Times = parseirq2(irq2in);
        char output[65536][9], (*out)[9]; // output is an array of all lines for the
memout. and out is a pointer to it
        char diskArr[65536][9], (*disk)[9] = diskArr; // diskArr contains the disk's
memory and i read like the input file the pointer is a portable array pointer for it
```

חלק זה פותח את קבצי הקלט לקריאה או כתיבה לפי הצורך ויוצא במקרה של שגיאה. בנוסף הוא קורא את הזכרון ופותח זכרון דינאמי לשורות.

```
        memin = openFile(argv[1], "r"); //input file opening
        trace = openFile(argv[6], "w");//trace file opening
        leds = openFile(argv[9], "w");//leds file opening
        display = openFile(argv[10], "w");//display file opening
        diskin = openFile(argv[2], "r");// open the disk input file
        hwregtrace = openFile(argv[7], "w");// open the hwregtrace file and exit on
fail
        out = memRead(output, memin); // read the memory
        disk = memRead(diskArr, diskin); // read the disk using same function
        MemoryLine* current = malloc(sizeof(MemoryLine)); // current line structure


        while (!feof(memin)) { // start reading input file line by line and process it
update the disk timer. this wil deal with irq1
```

חלק זה מבצע בדיקה האם יש לבצע פעולת irq מעתיק את הזכרון ל מערכך העזר current ומדפיס לקובץ הפלט trace. כלומר זהו חלק המטפל בirq.

```
                updateDiskTimer(ioRege);
                checkirq2(ioRege, count, irq2Times); // deal with irq2
                pc = checkirq(ioRege, pc, memin); // move pc in case of an irq event
                current = Create(memin, current); // copy line to struct
                Reg_Array[1] = current->imm; // copy immediate to register
                Print_To_Trace(trace, pc, current->line, Reg_Array);//print to trace
```

כעת נבצע את החלק האחראי לפקודות, ראשית נבדוק האם הפקודה היא איננה פקודת halt כלומר יציאה, אם התנאי מתקיים נבצע את פעולת הopcode נזוז לשורה הרצויה בmemin נעדכן את מחזור השעון ואת הטיימר. במקרה ואכן התקבלה פקודת halt נבצע אותה.

```
                if (current->opcode != 19) { // check if not on a halt
                        pc = Opcode_Operation(current, rege, ioRege, pc, memin, out,
hwregtrace, count, leds, display, disk); // perform opcode operation then go to correct
row.
```

```
                moveFP(memin, pc); // move file pointer to correct row
                count = updateclks(count, ioRege); //update clock cycles
                updateTimer(ioRege); // update the timer
            }
            else { // in case of a halt. just update clock cycle and leave
                count = updateclks(count, ioRege); //update clock cycles
                break;
            }
    }
```

לבסוף נבצע את הכתיבה לקבצי הפלט ונסגור ונשחרר את הקבצים והזכרונות הדינאמים

```
    //opening output files to be written after the run
    mem_out = openFile(argv[4], "w");
    regout = openFile(argv[5], "w");
    cycles = openFile(argv[8], "w");
    diskout = openFile(argv[11], "w");
    Print_To_Files(mem_out, regout, trace, cycles, output, count, Reg_Array, disk,
diskout); //print to files not written during run
    closeAndFree(memin, mem_out, regout, trace, hwregtrace, cycles, leds, display,
diskin, irq2in, diskout, irq2Times, current); // close all files
}
```

הסבר קצר על תכניות בדיקה:

כמו כן, על מנת לבדוק שהאסמבלר והסימולטור מתפקדים בצורה תקינה, נכתבו 6 תוכנות אסמבלי קצרות לכתיבה, כל התוכנות הורצו ללא שגיאה בשתי תוכנות הסי. הנה הסבר קצר על כל תכנית. אשר לה הערות בקובץ האסמבלי:

Summat: שומרת שני מערכים בתאים 0x100-0x10F, 0x110-0x11F ואז סוכמת אותם איבר איבר ושומרת את התוצאות ב0x120-0x12F

Bubble – מכניסה מערך לתאים 1024-1039 וממיינת אותו באלגוריתם bubblesort

Qsort - מכניסה מערך לתאים 1024-1039 וממיינת אותו באלגוריתם quicksort. בקוד זה ישנה המון רקורסיה וקריאה לפונקציות. והייתה השתדלות שהערכים אותן הן מקבלות יגיעו דרך רג'יסטרים a0 וa1 והערכים אותן הן מחזירות דרך v0

Leds – משתמשת בטיימר כדי להדליק כל נורה למשך שניה ואז להדליק את הנורה הבאה. נגמר אחרי שעברנו את כל הנורות

Clock – מפעילה את הצג כשעון. כאשר המספרים מוצגים בפורמט HH:MM:SS. משתמשת במעברים מיוחדים כדי שזה מה שיהיה כתוב על הצג כי הצג הקסאדצימלי

Disktest – מעתיקה את 4 הסקטורים הראשונים של הדיסק(128 שורות לכל סקטור) לזיכרון ומדביקה אותם ל4 הסקטורים הבאים שלו

כמו כן קיים קובץ בדיקה נוסף fib.asm אשר מחשב את איברי סדרת פיבונאצ'י. קובץ זה נכתב על ידי צוות הקורס ועזר לנו בבדיקות הראשונות. הוא נמצא בתיקיה tests רק לשם הוכחת עבודה. כאשר הקבצים אשר יש להם את הסיומת 2 הם קבצים שאנחנו יצרנו בהרצות. והקבצים שאין להם נתנו מצוות הקורס

עבור כל תכנית ישנם כל קבצי קלט ופלט של האסמבלר והסימולטור על מנת לקבל הוכחה שהפרוייקט עובד.