מגישים: רון מונדשיין 318895877 יניב מועלם 209127612 עדן דאיה 315285759

ספר פרויקט מבנה המחשב
ISA Project

תוכן עניינים

מבוא

במסגרת הקורס במבנה המחשב מימשנו מעבד SIMP, הדומה למעבד MIPS רק יותר פשוט.
מימשנו את ההוראות בפורמט מוגדר שיוזכר בהמשך, מימשנו אסמבלר וסימולטור בשפת C, התמודדנו עם התקני קלט פלט וכתבנו 4 קבצי אסמבלי כדי לבחון ולבדוק את עבודתנו.

רגיסטרים

כדי לממש את מעבד הSIMP נעזרנו ב16 רגיסטרים, כל אחד באורך 32 ביטים. שמם, המספר שלהם ותפקידם מובא בטבלה הבאה:

מספר רגיסטר	שם רגיסטר	מטרה
0	$zero	רגיסטר קבוע השומר את הערך 0
1	$imm1	שדה הimmediate הראשון לאחר sign extended
2	$imm2	שדה הimmediate השני לאחר sign extended
3	$v0	רגיסטר תוצאה
4	$a0	רגיסטר ארגומנט
5	$a1	
6	$a2	
7	$t0	רגיסטר למשתנה זמני
8	$t1	
9	$t2	
10	$s0	רגיסטר למשתנה שמור
11	$s1	
12	$s2	
13	$gp	רגיסטר למשתנה גלובלי
14	$sp	רגיסטר מחסנית
15	$ra	רגיסטר לכתובת חזרה לאחר שימוש בפונקציה

טבלה 1- ריגסטרים

הוראות

להוראות מעבד הMIPS יש רוחב של 48 ביטים ויכולות להיות עד 4096 שורות.
רגיסטר הPC באורך של 12 ביטים שמטרתו לייצג את 4096 השורות, כך שהייצוג נכנס בעד 12 ביטים.
במהלך המימוש ובהתאם להנחיות, מעבר בין הוראה לזו הבא אחריה מעלה את הPC ב1.

עבדנו עם פורמט הוראות אחיד לכל ההוראות, המיוצג בצורה הבא:

47:40	39:36	35:32	31:28	27:24	23:12	11:0
opcode	rd	rs	Rt	rm	Immediate1	Immediate2

טבלה 2-פורמט הוראות

Back to top

המעבד תומך ב21 ההוראות הבאות:

מספר opcode	שם	משמעות
0	Add	$R[rd] = R[rs] + R[rt] + R[rm]$
1	Sub	$R[rd] = R[rs] - R[rt] - R[rm]$
2	Mac	$R[rd] = R[rs] * R[rt] + R[rm]$
3	And	$R[rd] = R[rs] \& R[rt] \& R[rm]$
4	Or	$R[rd] = R[rs] \mid R[rt] \mid R[rm]$
5	Xor	$R[rd] = R[rs]\hat{}R[rt]\hat{}R[rm]$
6	Sll	$R[rd] = R[rs] \ll R[rt]$
7	Sra	$R[rd] = R[rs] \gg R[rt], arithmetic\ shift\ sign\ extension$
8	Srl	$R[rd] = R[rs] + R[rt] + R[rm], logical\ shift$
9	Beq	$if(R[rd] == R[rt]) \rightarrow pc = R[rm](low\ bits\ 11{:}0)$
10	Bne	$if(R[rd]! = R[rt]) \rightarrow pc = R[rm](low\ bits\ 11{:}0)$
11	Blt	$if(R[rd] < R[rt]) \rightarrow pc = R[rm](low\ bits\ 11{:}0)$
12	Bgt	$if(R[rd] > R[rt]) \rightarrow pc = R[rm](low\ bits\ 11{:}0)$
13	ble	$if(R[rd] <= R[rt]) \rightarrow pc = R[rm](low\ bits\ 11{:}0)$
14	bge	$if(R[rd] >= R[rt]) \rightarrow pc = R[rm](low\ bits\ 11{:}0)$
15	jal	$R[rd] = pc + 1, next\ instruction\ address, pc = R[rm][11{:}0]$
16	lw	$R[rd] = MEM\big[R[rs] + R[\,rt]\big] + R[rm]$
17	sw	$MEM\big[R[rs] + R[\,rt]\big] = R[rm] + R[rd]$
18	reti	$PC = IORegister[7]$
19	in	$R[rd] = IORegister[R[rs] + R[rt]]$
20	out	$IORegister\big[R[rs] + R[rt]\big] = R[rm]$
21	halt	$Halt\ execution, exit\ simulator$

טבלה 3- ההוראות

נשים לב כי כל פעולות הקפיצה השתמשנו ב12 LSB מכיוון שגודל הזיכרון הינו 4096 שורות.
כל שדה של immediate יכול להכיל 1 מ3 אפשרויות:
מספר דצימבלי חיובי או שלילי, מספר הקסדצימלי המתחיל ב $0x$ ולאחר מכן ספרות הקסדצימליות או LABEL, שם. סימבולי שמתחיל באות ומסתיים ב: ומייצג שורה בקוד.
אנו מניחים שכל פקודה לוקחת מחזור שעון בודד, כמצוין במטלה.

זיכרון DATA

לזיכרון הDATA יש רוחב של 32 ביטים ויכול להכיל עד 4096 שורות, לכן כמו ההגבלה על הPC, הכתובת גישה אליו יכולה להיות מיוצגת על ידי 12 ביטים.
דרך שבה אנחנו יכולים לאחסן מילה באורך 32 ביטים ישירות בזיכרון היא על ידי הפקודה $word$. ולאחר מכן ציון הערך. צורת שימוש: $word\ address\ data$.

Back to top

האסמבלר

asm.c

מטרת האסמבלר הינה לתכנת את המעבד באופן נוח וליצור תמונת זיכרון.
האסמבלר נכתב בשפת C וכלל את הספריות הבאות: stdlib.h , stdio.h, string.h.

האסמבלר יקבל כקלט תוכנית אסמבלי שכתובה כטקסט בשפת האסמבלי ויתרגם אותה לשפת מכונה. כפי שצוין במטלה הנחנו כי הקלט לאמסבלי תקין.
הקוד בנוי כך שאין הגבלה על כמות השורות שיתקבלו בקובץ הקלט, תחת ההנחות שיצויינו בהמשך.

האסמבלר יחזיר 2 קבצים:

1- Imemin.txt: תמונת ההוראות ההתחלתית. שורה הכתובה בספרות הקסאדצימליות באורך של 12 אותיות לכל שורה. הקובץ מקודד את כל ההוראות, שורה אחרי שורה תוך כדי קידוד הריגסטרים לפי טבלה 1, ה opcode לפי טבלה 3 ולפי הפורמט המפורט בטבלה 2.
2- Dmemin.txt: תמונת הזיכרון ההתחלתית. מוזנת מהפעלת הפקודה $word$. בקוד האסמבלי.

האסמבלר ירוץ על ידי הרצת השורה הבאה:
asm.exe program.asm imemin.txt dmemin.txt
כך שprogram הינה שם התוכנית השפת האסמבלי.

הנחות שבהן השתמשנו:

- אורך כל Label הוא עד 50 תווים.
- אורך שורה מקסימלית בקובץ הקלט היא 500 תווים.
- פורמט הLabel הינו אות גדולה ולאחר מכן אותיות או מספרים, לבסוף סימן נקודתיים ':'.
- ניתן להתעלם מרווחים, טאבים (/t), שורות ריקות או שורות שמכילות הערות בלבד.

תיאור מבני העזר השונים בהם השתמשנו:

- Label

נרצה לשמור כמות לא ידועה ולא מוגבלת של labels. בנוסף, נרצה לדעת לאיזה שורה לקפוץ בקוד כל פעם שמשתמש יציין את השם של label מסויים אליו הוא ירצה להגיע.
הגדרת המבנה עזר:

```
/*Struct that keeps Label data- name, it location in the assembly code and the next label*/
typedef struct Label {
    char name[50]; /*label's name, max size is 50*/
    char address[50]; /*label's location in the assembly code*/
    struct Label *next; /*next label in the list*/
} Label;
```

הגדרת הlabel כוללת את השם תחת ההנחות שציינו, המיקום בקוד ואת האיבר הבא, זאת מכיוון שאנחנו עובדים עם הlabels כרשימה מקושרת היות ואנו לא יודעים מראש את הכמות. הבחירה

Back to top

לעבוד בchar נעשתה מתוך נוחות, הקלט עצמו מתקבל כטקסט ולבסוף אנחנו מדפיסים טקסט ולכן נוכל לעבוד עם מערך של תווים ולחסוך המרות.

```
/*assign address to label*/
void assignAddress(struct Label *newLabel, int address) {
    sprintf(newLabel->address, "%d", address);
}
```

מבצעת השמה של כתובת לתוך מבנה נתונים של label, ממירה מספרות למערך של תווים לנוחות בהמשך.

```
/*
Add new label to the end of the labels list
input- head_Label, label's name and it's location in the code
output- none (void)
*/
void addLabel(Label **head, char *name, int address) {
    /*Make new label*/
    Label *newLabel;
    newLabel = (Label *) malloc(sizeof(Label));
    strcpy(newLabel->name, name);
    assignAddress(newLabel, address);
    newLabel->next = NULL;
    /*if no head yet, make the new label the head label*/
    if (*head == NULL) {
        *head = newLabel;
    } else {
        /*put the new label last in the list*/
        Label *current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newLabel;
    }
}
```

פונקציה שבונה את שרשרת החוליות, מוסיפה חוליה חדשה, אם היא הראשונה אז מגדירה אותה כראש, אחרת מוסיפה אותה לסוף.

```
/*
Find label location by it's name
input- head_Label, and label's name
```

Back to top

```
output label's location in the code
*/
char* findLabel(Label *head, char *name) {
    Label *current = head;
    while (current != NULL) {
        /*if the current node's label is the same as we want, return it address*/
        if (strcmp(current->name, name) == 0) {
            return current->address;
        }
        current = current->next;
    }
    /*No address to be found*/
    return NULL;
}
```

מקבלת את שרשרת החוליות שלהlabels ומחזירה את המיקום, בפונקציה נשתמש במעבר השני שנרצה להתאים בין שורת הקוד בה כתוב שם של label לבין השורה בו הlabel נמצא. אם לא מצאה תחזיר NULL.

```
/*
Destroy the labels list
input- head_Label
output- none (void)
*/
void destroy_labels(Label *head) {
    Label *current = head;
    while (current != NULL) {
        Label *temp = current;
        /*move to the next label*/
        current = current->next;
        /*free the current label*/
        free(temp);
    }
}
```

פונקציה שמשמשת למחיקת רשימת החוליות של הlabels ושחרור הזכרון.

- instructionMemory

מבנה נתונים ששומר את ההוראות של תוכנית האסמבלי, המקודדות כפי שמתואר בטבלה 2. נשתמש בשרשרת חוליות שוב כי אנחנו לא יודעים כמה שורות בדיוק יש בתוכנית וגם כדי לתת

Back to top

תמיכה בתוכנית באורך מאוד ארוך ("אינסופי").
הגדרת מבנה העזר:

```
/*instructionMemory struct contains: place of line of code, opcode, rd, rs, rt, imm1, imm2*/
typedef struct instructionMemory {
    int place; /*place of line of code*/
    char* opcode; /*opcode*/
    char* rd; /*register rd*/
    char* rs; /*register rs*/
    char* rt; /*register t*/
    char* rm; /*register rm*/
    char* imm1; /*imm1*/
    char* imm2; /*imm2*/
    struct instructionMemory *next; /*next memory line in the list*/
} instructionMemory;
```

מבנה עזר המכיל את השורה בקוד שההוראה נמצאת בו ואת כל חלקי ההוראה המתוארים בטבלה 2. כל חלק מההוראה נשמר כמערך של תווים. כמו"כ נשמרת ההוראה הבאה כדי לתחזק את מבנה שרשרת החוליות.

```
/*
Make new instructionMemory line
input: place of line of code, opcode, rd, rs, rt, imm1, imm2
output: new memory line
*/
instructionMemory *makeMemoryLine(int place, char opcode[20], char rd[20], char rs[20],
char rt[20], char rm[20], char imm1[20], char imm2[20], instructionMemory *next) {
    instructionMemory *newInsMemory = (instructionMemory *)
malloc(sizeof(instructionMemory));
    newInsMemory->place = place;
    newInsMemory->opcode = (char*)malloc(sizeof(char)*20);
    strcpy(newInsMemory->opcode, opcode);
    newInsMemory->rd = (char*)malloc(sizeof(char)*20);
    strcpy(newInsMemory->rd, rd);
    newInsMemory->rs = (char*)malloc(sizeof(char)*20);
    strcpy(newInsMemory->rs, rs);
    newInsMemory->rt = (char*)malloc(sizeof(char)*20);
    strcpy(newInsMemory->rt, rt);
    newInsMemory->rm = (char*)malloc(sizeof(char)*20);
    strcpy(newInsMemory->rm, rm);
```

Back to top

```
    newInsMemory->imm1 = (char*)malloc(sizeof(char)*20);
    strcpy(newInsMemory->imm1, imm1);
    newInsMemory->imm2 = (char*)malloc(sizeof(char)*20);
    strcpy(newInsMemory->imm2, imm2);
    newInsMemory->next = next;
    return newInsMemory;
}
```

פונקציה שבונה הוראה ומחזירה אותה, על ידי כך שהיא מעתיקה את כל כל הרכיבים.

```
/*
Add new instructionMemory to the end of the memory lines list
input- head_instructionMemory, place of line of code, opcode, rd, rs, rt, imm1, imm2
output- none (void)
*/
void addInsMemory(instructionMemory **head, int place, char* opcode, char* rd, char* rs,
char* rt, char* rm, char* imm1, char* imm2) {
    /*Make new memory line*/
    instructionMemory *newMemoryLine = makeMemoryLine(place, opcode, rd, rs, rt, rm,
imm1, imm2, NULL);
    /*if no head yet, make the new instruction memory line the head memory line*/
    if (*head == NULL) {
        *head = newMemoryLine;
    } else {
        /*put the new instruction memory line last in the list*/
        instructionMemory *current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newMemoryLine;
    }
}
```

פונקציה שמוסיפה הוראה לשרשרת החוליות, אם השרשרת ריקה, מגדירה אותה כראש. אחרת מוסיפה אותה לסוף.

```
/*
Destroy the instruction memory lines list
input- head_instructionMemory
output- none (void)
*/
```

Back to top

```
void destroy_memory_lines(instructionMemory *head) {
    instructionMemory *current = head;
    while (current != NULL) {
        instructionMemory *temp = current;
        /*move to the next memory line*/
        current = current->next;
        /*free the current memory line*/
        free(temp);
    }
}
```

פונקציה שהורסת הוראה, נעזר בה לשחרור הזיכרון בתום התוכנית.

- dataMemory

מבנה נתונים שנעזר בו כדי לשמור על DATA שנרצה להכניס לזיכרון, לבסוף יופיע בקובץ Dmemin.txt. כותב קוד האסמבלי יכול להוסיף DATA לזיכרון על ידי שימוש בפונקציית $word$. שתוארה בחלק הקודם.

שומר את המיקום של הערך רצוי בתמונת הזיכרון ואת הערך עצמו.

```
/*Data memory struct contains: place, value*/
typedef struct dataMemory {
    int place; /*place of line of code*/
    int value; /*value*/
    struct dataMemory *next; /*next data memory line in the list*/
} dataMemory;
```

מבנה עזר המכיל את המיקום של הערך בתמונת הזיכרון, הערך והערך הבא. נבחר להשתמש במבנה נתונים בשביל הנוחות למרות שגודל הזיכרון ידוע.

```
/*
Make data memory  line
input: place of line of code, value
output: new memory line
*/
dataMemory *makeDataMemoryLine(int place, int value, dataMemory *next) {
    dataMemory *newDataMemory = (dataMemory *) malloc(sizeof(dataMemory));
    newDataMemory->place = place;
    newDataMemory->value = value;
    newDataMemory->next = next;
    return newDataMemory;
}
```

פונקציה שבונה מבנה נתונים של שורה בזיכרון.

Back to top

```c
/*
Add new dataMemory to the end of the memory lines list
input- head_dataMemory, place of line of code, opcode, rd, rs, rt, imm1, imm2
output- none (void)
*/
void addDataMemory(dataMemory **head, int place, int value) {
    /*Make new memory line*/
    dataMemory *newDataMemory = makeDataMemoryLine(place, value, NULL);
    /*if no head yet, make the new data memory line the head memory line*/
    if (*head == NULL) {
        *head = newDataMemory;
    } else {
        /*put the new data memory line last in the list*/
        dataMemory *current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newDataMemory;
    }
}
```

מוסיפה את ההוראה לשרשרת החוליות של ההוראות, מוסיפה לסוף או להתחלה (במידה והשרשרת ריקה).

```c
/*
Destroy the instruction memory lines list
input- head_instructionMemory
output- none (void)
*/
void destroy_data_memory(dataMemory *head) {
    dataMemory *current = head;
    while (current != NULL) {
        dataMemory *temp = current;
        /*move to the next memory line*/
        current = current->next;
        /*free the current memory line*/
        free(temp);
    }
}
```

Back to top

פונקציה שהורסת הוראה, נעזר בה לשחרור הזיכרון בתום התוכנית.

- totalMemory

מבנה נתונים שמטרתו להחזיר 2 מבני נתונים שונים כאשר יוצאים מפונקציה (return).

```
/*
Total memory struct contains: instruction memory and data memory
*/
typedef struct totalMemory {
    instructionMemory *head_instructionMemory; /*head of instruction memory*/
    dataMemory *head_dataMemory; /*head of data memory*/
} totalMemory;
```

מכילה את הראש של שרשרת החוליות של ההוראות והראש של שרשרת החוליות של הזיכרון DATA.

תיאור פונקציות העזר:

```
/*
Returns if there is a function in the line
input- line
output- 1 if there is a function, 0 if not
*/
int there_is_a_function(char* line){
    int there_is_a_function = 0;
    for (int i = 0; line[i] != '\0'; i++) {
        if (line[i] == '#') break;
        if (line[i] == ':') continue;
        if (line[i] == '$') {
            there_is_a_function = 1;
            break;
        }
    }
    return there_is_a_function;
}
```

פונקציה שבודקת האם בשורת קוד נתונה יש פונקציה. אם יש הערה שמתחילה לפני שזיהינו פונקציה נחזיר כי אין, אם נראה : שמסמנות על label נמשיך לחפש כי תתכן label והוראה באותה שורה. אם נזהה $ נחזיר כי יש פונקציה. תחת ההנחות המצויות בשאלה.

```
/*
Returns if there is a label in the line, if there is, return the place of :
input- line
```

Back to top

```
output- the place of : if there is a label, -1 if not
*/
int there_is_a_label(char* line){
    int there_is_a_label;
    there_is_a_label = -1;
    for (int i = 0; i < sizeof(line); i++)
    {
        if (line[i] == '#')
        {
            there_is_a_label = -1;
            break;
        }
        if (line[i] == ':')
        {
            there_is_a_label = i;
            break;
        }
    }

    return there_is_a_label;
}
```

פונקציה שבודקת האם יש label בשורה. אם זיהינו הערה לפני שנזהה label נחזיר כי אין. אחרת אם זיהינו : נחזיר כי יש label. אם אין נחזיר -1 ואחרת נחזיר מספר אי שלילי כלשהו שמציין מיקום.

```
/*
Help function that find out if a char is a digit
input- char
output- 1 if it is a digit, 0 if not
*/
int charIsDigit(char c) {
    /* Check if the character is a digit (0-9)*/
    if (c >= '0' && c <= '9') {
        return 1;  /* True (character is a digit)*/
    } else {
        return 0;  /* False (character is not a digit) */
    }
}
```

Back to top

פונקציית עזר שממירה מתו לספרה.

```
/*
Help function that find out if a char is a letter
input- char
output- 1 if it is a letter, 0 if not
*/
int charIsLetter(char c) {
    /* Check if the character is a letter (a-z, A-Z)*/
    if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) {
        return 1;  /* True (character is a letter)*/
    } else {
        return 0;  /* False (character is not a letter) */
    }
}
```

פונקציית עזר שמחזירה האם התו שהתקבל הוא תו בא"ב האנגלי, גדול או קטן.

```
/*
from string to number
input- string
output- int (the decimal number), -1 if there is an error
*/
int fromStringToNumber(char* string){
    int number;
    number = 0;
    for (int i = 0; i < strlen(string); i++)
    {
        /*is hex number*/
        if (string[0] ==  '0' && string[1] == 'x')
        {
            if (i > 1)
            {
                if ((string[i] - '0' >= 0) && (string[i] - '0' < 10))
                {
                    number = number * 16 + (string[i] - '0');
                }
                else if ((string[i] - 'A' >= 0) && (string[i] - 'A' < 6))
```

Back to top

```
            {
                number = number * 16 + (string[i] - 'A' + 10);
            }
            else if ((string[i] - 'a' >= 0) && (string[i] - 'a' < 6))
            {
                number = number * 16 + (string[i] - 'a' + 10);
            }
        }
    }
    /*is negetive number*/
    else if (string[0] ==  '-')
    {
        if (i > 0 )
        {
            number = number * 10 + (string[i] - '0');
        }
    }
    else /*is positive number*/
    {
        number = number * 10 + (string[i] - '0');
    }
}
if (string[0] == '-')
{
    number = -number;
}
return number;
}
```

פונקציה שממירה ממערך של תווים (string) למספר, מחולקת ל3 קטעים. בחלק הראשון אם מתחיל ב0x מזהה כי מדובר המספר האקסדצימלי וממירה בהתאם לבסיס עשרוני.
בחלק. השני מזהה כי מדובר במספר עשרוני שלילי ומחזירה מספר עשרוני שלילי. בחלק השלישי מחזירה את המספר כמו שהוא, עשרוני חיובי. פעלנו כך כדי לתמוך ב3 הצורות של הערכים המספריים שאנחנו יכולים לקבל כמתואר במטלה.

```
/*
help function that copy a section from a string from place i to place j to new string
input- string, i, j
output- new string
```

Back to top

```
*/
char* copySection(char* string, int i, int j){
    char* newString;
    int k;
    k = 0;
    newString = (char*)malloc(sizeof(char)*(j-i+1));
    for (int place = i; place < j; place++)
    {
        newString[k] = string[place];
        k++;
    }
    newString[k] = '\0';
    return newString;
}
```

פונקציה שמקבלת מצביע לתו, אינדקס התחלה ואינדקס סוף ומחזירה את התווים שנמצאים בקטע הזה.
נשתמש בפונקציה זו בחלק של המעבר השני על קוד האסמבלר אותו נתאר בהמשך.

פונקציות הדפסה לקבצים ופונקציות העזר של ההדפסה

```
/*
from opcode to hex
input- opcode in char*
output- opcode in hex
*/
char* from_opcode_to_hex(char* opcode){
    char* hex_opcode;
    if (strcmp(opcode, "add") == 0) return "00";
    else if (strcmp(opcode, "sub") == 0) return "01";
    else if (strcmp(opcode, "mac") == 0) return "02";
    else if (strcmp(opcode, "and") == 0) return "03";
    else if (strcmp(opcode, "or") == 0) return "04";
    else if (strcmp(opcode, "xor") == 0) return "05";
    else if (strcmp(opcode, "sll") == 0) return "06";
    else if (strcmp(opcode, "sra") == 0) return "07";
    else if (strcmp(opcode, "srl") == 0) return "08";
    else if (strcmp(opcode, "beq") == 0) return "09";
    else if (strcmp(opcode, "bne") == 0) return "0A";
    else if (strcmp(opcode, "blt") == 0) return "0B";
```

Back to top

```
    else if (strcmp(opcode, "bgt") == 0) return "0C";
    else if (strcmp(opcode, "ble") == 0) return "0D";
    else if (strcmp(opcode, "bge") == 0) return "0E";
    else if (strcmp(opcode, "jal") == 0) return "0F";
    else if (strcmp(opcode, "lw") == 0) return "10";
    else if (strcmp(opcode, "sw") == 0) return "11";
    else if (strcmp(opcode, "reti") == 0) return "12";
    else if (strcmp(opcode, "in") == 0) return "13";
    else if (strcmp(opcode, "out") == 0) return "14";
    else /*halt*/ return "15";
}
```

פונקציה שממירה מopcode לערך האקסדצימלי שלו.

```
/*
from register to hex
input- register in char
output- register in hex
*/
char from_register_to_hex(char* opcode){
    if (strcmp(opcode, "$zero") == 0) return '0';
    else if (strcmp(opcode, "$imm1") == 0) return '1';
    else if (strcmp(opcode, "$imm2") == 0) return '2';
    else if (strcmp(opcode, "$v0") == 0) return '3';
    else if (strcmp(opcode, "$a0") == 0) return '4';
    else if (strcmp(opcode, "$a1") == 0) return '5';
    else if (strcmp(opcode, "$a2") == 0) return '6';
    else if (strcmp(opcode, "$t0") == 0) return '7';
    else if (strcmp(opcode, "$t1") == 0) return '8';
    else if (strcmp(opcode, "$t2") == 0) return '9';
    else if (strcmp(opcode, "$s0") == 0) return 'A';
    else if (strcmp(opcode, "$s1") == 0) return 'B';
    else if (strcmp(opcode, "$s2") == 0) return 'C';
    else if (strcmp(opcode, "$gp") == 0) return 'D';
    else if (strcmp(opcode, "$sp") == 0) return 'E';
    else /*ra*/ return 'F';
}
```

פונקציה שממירה מהרגיסטר לערך האקסדצימלי שלו.

```
/*
```

Back to top

```
from decimal to 3 hex digits
input- decimal number in char*
output- 3 hex digits
*/
char* from_decimal_to_3_hex_digits(char* number){
    int number_int;
    if (number[0] == '-')
    {
        number_int = fromStringToNumber(number + 1); /* get the absolute value */
        number_int = (1 << 12) - number_int; /* calculate 2's complement */
    } else{
        number_int = fromStringToNumber(number);
    }
    char* hex;
    hex = (char*)malloc(sizeof(char)*4); /* allocate 4 chars for 3 hex digits and the null
terminator */
    sprintf(hex, "%03X", number_int);
    return hex;
}
```

פונקציה שממירה ממספר דצימלי למספר אקסדצימלי בעל 3 ספרות אקסדצימליות.

```
/*
from int decimal to 8 char hex
input- decimal number
output- 8 char hex
*/
char* from_decimal_to_8_char_hex(int number){
    char* hex;
    hex = (char*)malloc(sizeof(char)*7); /* allocate 7 chars for 6 hex digits and the null
terminator */
    sprintf(hex, "%08X", number);
    return hex;
}
```

פונקציה שממירה ממספר דצימלי לאקסדצימלי בפורמט של 8 ספרות אקסדצימליות.

```
/*
Save the Instructions memory in the output file named inemin.txt"
input- total_memory
output- none (void)
```

Back to top

```
*/
void save_instruction_memory_into_file(totalMemory *total, Label *head_Label, char*
output_file)
{
    /*Make new file*/
    FILE *file;
    file = fopen(output_file, "w");
    if (file == NULL) {
        printf("Error opening file\n");
        exit(1);
    }
    /*Make the current instruction memory line*/
    instructionMemory *current_instruction_memory_line = total->head_instructionMemory;
    /*Save the instruction memory lines in the file*/
    while (current_instruction_memory_line != NULL)
    {
        if(findLabel(head_Label, current_instruction_memory_line->imm1) != NULL){
            current_instruction_memory_line->imm1 = findLabel(head_Label,
current_instruction_memory_line->imm1);
        }
        if(findLabel(head_Label, current_instruction_memory_line->imm2) != NULL){
            current_instruction_memory_line->imm2 = findLabel(head_Label,
current_instruction_memory_line->imm2);
        }

        char* decimal_to_hex1 =
from_decimal_to_3_hex_digits(current_instruction_memory_line->imm1);
        char* decimal_to_hex2 =
from_decimal_to_3_hex_digits(current_instruction_memory_line->imm2);
        fprintf(file, "%s%c%c%c%c%s%s\n",
from_opcode_to_hex(current_instruction_memory_line->opcode),
from_register_to_hex(current_instruction_memory_line->rd),
from_register_to_hex(current_instruction_memory_line->rs),
from_register_to_hex(current_instruction_memory_line->rt),
from_register_to_hex(current_instruction_memory_line->rm),
from_decimal_to_3_hex_digits(current_instruction_memory_line->imm1),
from_decimal_to_3_hex_digits(current_instruction_memory_line->imm2));
```

Back to top

```
        current_instruction_memory_line = current_instruction_memory_line->next;
    }
}
```

פונקציה שמדפיסה לקובץ את כל שרשרת החוליות של הוראות התוכנית. תחילה היא פותחת קובץ חדש בשם אותו נקבל כקלט לתוכנית שלנו. לאחר מכן היא מתחילה מראש השרשרת, ההוראה הראשונה, ועוברת על כל ההוראות בצורה הבאה: אם יש label, מחליפה את ערך imm1 במיקום הlabel, בדומה עבור imm2. לאחר מכן משתמשת בפונקציות שממירות מדצימלי לאקסדצימלי בפורמט של 3 ספרות כדי להתאים לפורמט הרצוי, ביחד עם הפונקציה שמתאימה לרגיסטר את הערך האקסדצימלי שלו ולopcode את הערך האקסדצימלי שלו כותבת לקובץ את השורה המתוארת בטבלה 2 מומרת לשורה בעלת 12 ספרות אקסדצימליות כנדרש במטלה.

```
/*
Save the Data memory in the output file
input- total_memory
output- none (void)
*/
void sava_data_memory_into_file(totalMemory total, char* output_file)
{
    int max_place;
    max_place = 0;
    /*Make new file*/
    FILE *file;
    file = fopen(output_file, "w");
    if (file == NULL) {
        printf("Error opening file\n");
        exit(1);
    }
    /*Make the current data memory line*/
    dataMemory *current_data_memory_line = total.head_dataMemory;
    /*find most bigger place of a data memory*/

    while (current_data_memory_line != NULL)
    {
        if (current_data_memory_line->place > max_place)
        {
            max_place = current_data_memory_line->place;
        }
        current_data_memory_line = current_data_memory_line->next;
```

Back to top

```
    }
    current_data_memory_line = total.head_dataMemory;
    /*from data memory to array*/
    int data_memory_array[max_place];
    int i;
    i = 0;
    /*initilaze the array in zero*/
    for (int i = 0; i < max_place; i++)
    {
        data_memory_array[i] = 0;
    }
    /*Save the data memory lines in the file*/
    while (current_data_memory_line != NULL)
    {
        data_memory_array[current_data_memory_line->place] = current_data_memory_line-
>value;
        current_data_memory_line = current_data_memory_line->next;
    }
    /*Save the data memory array into file*/
    for (int i = 0; i < max_place + 1; i++)
    {
        fprintf(file, "%s\n", from_decimal_to_8_char_hex(data_memory_array[i]));
    }
}
```

פונקציה ששומרת את הDATA לקובץ המתאים, ששמו ניתן כקלט. הפונקציה פותחת את הקובץ, וראשית מוצאת מה המיקום הגדול ביותר של שורה בזיכרון בו נמצא מידע שונה מאפס. פעלנו תחת ההנחה שהוזכרה רבות בשיעור שאם לא הוכנס מידע לשורה מסוימת, ערכה הוא אפס וכך שאם סוף הזיכרון מורכב רק מאפסים לא צריך להדפיס אותו, נדפיס את כל השורות עד לשורה האחרונה ששונה מאפס. נאתחל מערך עד גודל אותו ערך מקסימלי שמצאנו. נאתחל אותו באפסים. לאחר מכן נוסיף את המידע שאספנו לאורך הקוד לפי מיקום הערך בזיכרון והערך עצמו.
נעבור בלולאה על המערך ונדפיס לקובץ בפורמט הרצוי בעזרת פונקציית עזר.

מעבר ראשון

```
/*
First pass on the input txt- find the labels and save them in the labels list
input- input txt file
output- head_Label
*/
```

Back to top

```
Label* first_pass(char* input_file){
    /*Open the input file*/
    FILE *file;
    file = fopen(input_file, "r");
    if (file == NULL) {
        printf("Error opening file\n");
        exit(1);
    }
    /*Make the head label*/
    Label *head_Label = NULL;
    /*Make the current line, max is 500*/
    char current_line[500];
    /*Make the current place*/
    int current_place = 0;
    /*Make the current label*/
    char* current_label_name;
    /*Make the current label's address*/
    int current_label_address;
    /*Save if found label- found - 1 not found - 0*/
    int found_label;
    int place_in_word;

    current_place = 0;
    found_label = 0;
    place_in_word = 0;
    strcpy(current_label_name, "");

    while (fgets(current_line, sizeof(current_line), file) != NULL)
    {
        if (current_line[0] == '\n' || current_line[0] == '#'  || current_line[0] == '\t' ||
        (current_line[0] == '.' && current_line[1] == 'w' && current_line[2] == 'o' &&
current_line[3] == 'r' && current_line[4] == 'd'))
        {
            continue;
        }
        for (int place = 0; place < sizeof(current_line); place++){
```

Back to top

```
        if (there_is_a_function(current_line) == 1) current_place++;

        /*if the line is a label*/
        if(current_line[place] == ':')
        {
            current_label_name = malloc(50);
            /* save label name*/
            for (int i = 0; i < place; i++)
            {
                if (current_label_name[i] == ' ' || current_label_name[i] == '\t' ||
current_label_name[i] == '\n') continue;

                current_label_name[place_in_word] = current_line[i];
                place_in_word++;
                found_label = 1;
            }
        }
        if (found_label == 1)
        {
            /* add label to the list*/
            addLabel(&head_Label, current_label_name, current_place);
            /*reset the variables*/
            free(current_label_name);
            found_label = 0;
            place_in_word = 0;
        }
    }
  }

  /* Close the file */
  fclose(file);
  /*return the labels list*/
  return head_Label;
}
```

פותח את הקובץ של הקוד במצב קריאה.
אם זו שורה ריקה או שורה שיש בה רק הערה אז אין בה label עובר מבלי לספור.
אם זו שורה שיש בה הוראה, סופר אותה.

Back to top

אם מזהה label, מעתיק את השם שלה אות, אות.
בונה label חדש, מוסיף לשרשרת החוליות.
משחרר זיכרון מיותר, סוגר את הקובץ ומסיים את הפונקציה.
כפי שניתן לראות המעבר הראשון מתעסק בlabel ומטרתו לבנות את שרשרת החוליות שתתאים שם של label עם השורה אליה נרצה לקפוץ אם נתקל בשם של הlabel כחלק מפקודה אחרת. נשתמש בשרשרת חוליות של הlabel שבנינו במעבר השני וכמובן שגם בהדפסות לקובץ, שם נדפיס את ערך הlabel ולא את שמו.

מעבר שני

```
/*
Second pass on the input txt- find the commands and save them in the memory lines list
input- input txt file
output- head_MemoryLine
*/
totalMemory* second_pass(char* input_file){
    /*Open the input file*/
    FILE *file;
    file = fopen(input_file, "r");
    if (file == NULL) {
        printf("Error opening file\n");
        exit(1);
    }
    /*Make the head memory line*/
    instructionMemory *head_MemoryLine = NULL;
    /*Make the current line, max is 500*/
    char current_line[500];
    /*Make the current place*/
    int current_place;
    /*Make the current opcode*/
    char current_opcode[20];
    /*Make the current rd*/
    char current_rd[20];
    /*Make the current rs*/
    char current_rs[20];
    /*Make the current rt*/
    char current_rt[20];
    /*Make the current rm*/
```

Back to top

```
char current_rm[20];
/*Make the current imm1*/
char current_imm1[20];
/*Make the current imm2*/
char current_imm2[20];
/*Make the data memory head*/
dataMemory *head_dataMemory = NULL;
char* number_char;
int number_int;
char* reg;
int label_place;
label_place = -1;
int begin;
int end;
int place;
int value;
place = 0;
value = 0;
begin = 0;
end = 0;

char dotword[6];
strcpy(dotword, ".word");
char *start;
start = &current_line[0];

totalMemory *total_memory = (totalMemory*)malloc(sizeof(totalMemory));

/*Save the current place*/
current_place = 0;
while (fgets(current_line, sizeof(current_line), file) != NULL) {
    label_place = there_is_a_label(current_line);
if (*current_line == '\n' || current_line[0] == '#') continue;
while (*start == ' ' || *start == '\t') { start++;}

if (strncmp(start, ".word", 5) == 0) {
        begin = 5;
```

Back to top

```
        sscanf(start, ".word %i %i", &current_place, &value);
        addDataMemory(&head_dataMemory, current_place, value);
        current_place++;
    }
    if (there_is_a_function(current_line) == 1) {
        /* save in instructionMemory */
        begin = (label_place != -1) ? label_place + 1 : (current_line[0] == '\t') ? 1 : 0;
        end = begin;
        while (current_line[end] != ' ') end++;
        /*saves opcode*/
        reg = copySection(current_line, begin, end);
        strcpy(current_opcode, reg);
        free(reg);
        begin = end + 1;
        /*checks the place of register rd*/
        while (current_line[begin] != '$') begin++;
        end = begin;
        while (current_line[end] != ',') end++;
        /*saves register rd*/
        reg = copySection(current_line, begin, end);
        strcpy(current_rd, reg);
        free(reg);
        begin = end + 1;
        /*checks the place of register rs*/
        while (current_line[begin] != '$') begin++;
        end = begin;
        while (current_line[end] != ',') end++;
        /*saves register rs*/
        reg = copySection(current_line, begin, end);
        strcpy(current_rs, reg);
        free(reg);
        begin = end + 1;
        /*checks the place of register rt*/
        while (current_line[begin] != '$') begin++;
        end = begin;
        while (current_line[end] != ',') end++;
        /*saves register rt*/
```

Back to top

```
        reg = copySection(current_line, begin, end);
        strcpy(current_rt, reg);
        free(reg);
        begin = end + 1;
        /*checks the place of register rm*/
        while (current_line[begin] != '$') begin++;
        end = begin;
        while (current_line[end] != ',') end++;
        /*saves register rm*/
        reg = copySection(current_line, begin, end);
        strcpy(current_rm, reg);
        free(reg);
        begin = end + 1;
        /*checks the place of imm1*/
        while (charIsDigit(current_line[begin]) != 1 && charIsLetter(current_line[begin]) != 1)
begin++;
        end = begin;
        while (current_line[end] != ',') end++;
        /*saves imm1*/
        if(current_line[begin - 1] == '-') begin--;
        reg = copySection(current_line, begin, end);
        strcpy(current_imm1, reg);
        free(reg);
        begin = end + 1;
        /*checks the place of imm2*/
        while (charIsDigit(current_line[begin]) != 1 && charIsLetter(current_line[begin]) != 1)
begin++;
        end = begin;
        while (current_line[end] != '\n' && current_line[end] != '\0' && current_line[end] != '#'
&& current_line[end] != ' ' && current_line[end] != '\t') end++;
        /*saves imm2*/
        if(current_line[begin - 1] == '-') begin--;
        reg = copySection(current_line, begin, end);
        strcpy(current_imm2, reg);
        free(reg);
        /*add to memory line*/
```

Back to top

```
            addInsMemory(&head_MemoryLine, current_place, current_opcode, current_rd,
current_rs, current_rt, current_rm, current_imm1, current_imm2);
        }
        current_place++;
    }
    /* Close the file */
    fclose(file);
    /*return total memory AKA data memory + instructions memory*/
    total_memory->head_instructionMemory = head_MemoryLine;
    total_memory->head_dataMemory = head_dataMemory;
    return total_memory;
}
```

הפונקציה עוברת על שורות קוד האסמבלי, אם זו שורה ריקה או שורה שמכילה הערה ממשיכה לשורה הבאה. אחרת, בודקת אם זאת פקודת word., אם כן ממירה את הערכים לint ומוסיפה חוליה חדשה לשרשרת החוליות של מבנה הDATA.

אם מצאנו פקודה (פונקציה) בשורת האסמבלי, נתחיל לפרק אותה גורם אחר גורם לפי המצויין בטבלה 2. המעבר נעשה "ידנית", זאת אומרת תו אחר תו עד לזיהוי סימן רלוונטי, למשל $, כדי שלא נפספס אף פורמט בו הקלט שלנו יכול להגיע. לכל קטע שכזה נמצא את ההתחלה והסוף של הגורם מטבלה 2 אותו אנחנו מחפשים, נעתיק אותו בעזרת פונקציית עזר שכתבנו, נשחרר זיכרון מיותר ונמשיך הלאה. לאחר שסיימנו עם כל הרכיבים נבנה רכיב חדש של מבנה הנתונים מסוג הוראה שתיארנו ונוסיף את ההוראה לשרשרת החוליות.

נסגור את הקובץ.

נאחד את 2 הראשים של מבני הנתונים שיצרנו, מבנה ההוראות ומבנה הDATA למבנה נתונים אחד, כדי לעמוד בתנאים של שפת התכנות C בה יכולים להחזיר רק מבנה נתונים יחיד, נחזיר אותו ונסיים.

במעבר זה נעשה התרגום מאסמבלי לשפת מכונה תוך כדי כך שאנחנו נעזרים בשרשרת החוליות של הlabel מהמעבר הראשון.

Main

```
/*
main function
*/
int main(int argc, char *argv[]) {
    Label *head_Label;
    totalMemory *total_memory;
    /*Check if the input is valid*/
    if (argc != 4) {
        printf("Error: invalid input\n");
        exit(1);
    }
```

Back to top

```
    /*Make the head label*/
    head_Label = first_pass(argv[1]);
    /*Make the total memory*/
    total_memory = second_pass(argv[1]);
    /*Save the data memory into file*/
    sava_data_memory_into_file(*total_memory, argv[3]);
    /*Save the instruction memory into file*/
    save_instruction_memory_into_file(total_memory, head_Label, argv[2]);

    /*free*/
    /*Destroy the labels list*/
    destroy_labels(head_Label);
    /*Destroy the data memory list*/
    destroy_data_memory(total_memory->head_dataMemory);
    /*Destroy the instruction memory list*/
    destroy_memory_lines(total_memory->head_instructionMemory);
    /*Destroy the total memory*/
    free(total_memory);
    return 0;
}
```

"לב" התוכנית- בודקת אם כמות הקלטים היא הכמות הרצויה, אם לא מחזירה שגיאה.
מריצה את המעבר הראשון ומחזירה שרשרת חוליות של label.
מריצה את המעבר השני ומחזירה את מבנה הנתונים אשר מאחד את מבנה הנתונים של ההוראות של התוכנית ומבנה הנתונים של הDATA שהמשתמש שומר בזיכרון בעזרת פקודת ***word***..
מפעיל את 2 הפונקציות שכותבות את המידע לתוך קבצים מתאימים.
משחרר כל זיכרון שהוקצה לטובת התוכנית.
יוצא מן התוכנית בהצלחה!

הסימולטור

sim.c

מטרת העל של הסימולטור הוא כשמו, לבצע סימולציה לקוד כאילו הוא מעבד שקורה קוד בשפת מכונה ומבצע את הפעולות המתאימות.
הסימולטור נכתב בשפת C וכלל את הספריות הבאות: stdlib.h , stdio.h, string.h.
הסימולטור מסמלץ את לולאת הfetch-decode-execute.
אנחנו מתחילים משורה מספר אפס, PC=0.
בכל איטרציה אנחנו מפענחים הוראה, מבצעים אותה בהתאם לקידוד שתיארנו בחלק המבוא, מעדכנים רגיסטרים מתאימים ובסוף ההרצה קופצים להוראה הבאה. ההוראה הבאה יכולה להיות בPC=PC+1, אך בהוראות קפיצה נקפוץ בהתאם להוראה. כל הקפיצות נעשות בהתאמה לכך שיכולות להיות עד4096 שורות

Back to top

הוראה, המיוצגות על ידי 12 ביטים לכן אם במסגרת החישוב חרגנו מכמות הביטים, ניקח את 12 הביטים הLSB.
במהלך
הסימולטור יסתיים עם קבלת הוראת HALT, או במידה והשורה האחרונה בתוכנית הסתיימה ולא כללה קפיצה.
הסימולטור ירוץ על ידי xhnהרצת השורה הבאה:

sim.exe imemin.txt dmemin.txt diskin.txt irq2in.txt dmemout.txt regout.txt trace.txt hwregtrace.txt cycles.txt leds.txt display7seg.txt diskout.txt monitor.txt monitor.yuv

כך שimemin.txt dmemin.txt הם הקבצים של תמונת הזיכרון הוראות ותמונת זיכרון הDATA, הפלטים של האסמבלר.
שאר הקבצים הם קבצי הפלט שתפקידם לתעד את הדרך בה רץ הקוד מכיוונים שונים.

תיאור מבני הנתונים בהם השתמשנו:

מערך זיכרון - לייצוג הזיכרון השתמשנו במערך בגודל 2^12 לייצוג 4096 שורות.
מערך דיסק - לייצוג הדיסק השתמשנו במערך בגודל 2^14 לייצוג 16kb מילים, שכן סך כל הדיסק הוא 64kb וגודל כל מילה הוא 4 בתים.
מערך מסך - לייצוג המסך השתמשנו במערך בגודל 2^16 לייצוג 256*256 פיקסלים.

מערך רגיסטרים – מערך בגודל 16 לייצוג הרגיסטרים הרגילים.
מערך רגיסטרי IO – מערך בגודל 23 לייצוג רגיסטרי IO.

כעת נציג את הפונקציות העיקריות והחשובות בסימולטור.

פונקציות כלליות

read_file_to_array – קוראת מידע מקובץ input ומכניסה אותו למערך.

```
/*read data from input files and put them in array
input - line_length, file
output - array of lines
*/
char **read_file_to_array(int line_length, FILE *file, size_t* line_amount) {
    char **lines = NULL;  // Array to store lines
    size_t line_count = 0;
    char buffer[line_length + 1];  // Buffer to store the current line

    while (fgets(buffer, sizeof(buffer), file) != NULL) {
        // Remove newline character at the end of the buffer, if present
        buffer[strcspn(buffer, "\n")] = '\0';
```

Back to top

```c
    // Skip empty lines or lines with only newline character
    if (strlen(buffer) == 0) {
        continue;
    }
        // Skip empty lines or lines with only whitespace characters
    if (is_whitespace(buffer)) {
        continue;
    }

    // Allocate memory for the new line pointer array
    char **temp_lines = realloc(lines, (line_count + 1) * sizeof(char *));
    if (temp_lines == NULL) {
        printf("Memory allocation error\n");
        // Free allocated memory for lines array
        for (size_t i = 0; i < line_count; ++i) {
            free(lines[i]);
        }
        free(lines);
        return NULL;
    }
    lines = temp_lines;

    // Allocate memory for the current line and copy the content
    lines[line_count] = malloc(strlen(buffer) + 1);
    if (lines[line_count] == NULL) {
        printf("Memory allocation error\n");
        // Free allocated memory for lines and lines array
        for (size_t i = 0; i < line_count; ++i) {
            free(lines[i]);
        }
        free(lines);
        return NULL;
    }
    strcpy(lines[line_count], buffer);

    // Increment the line count
    line_count++;
}

// Allocate space for the NULL terminator in the lines array
char **temp_lines = realloc(lines, (line_count + 1) * sizeof(char *));
```

Back to top

```
    if (temp_lines == NULL) {
        // Free allocated memory for lines array
        for (size_t i = 0; i < line_count; ++i) {
            free(lines[i]);
        }
        free(lines);
        printf("Memory allocation error for NULL terminator\n");
        return NULL;
    }
    lines = temp_lines;
    lines[line_count] = NULL;  // Null-terminate the lines array

    // Set the actual number of lines read
    *line_amount = line_count;

    return lines;
}
```

פונקציות המרה

ישנן מספר פונקציות להמרה בין בסיסים, שכן קלט ופלט מרבית הקבצים הם בהקסה, ובמהלך הריצה יש צורך לבצע המרות נדרשות. נציג את חלקן.

twos_complement – ממירה מהקסה לדצימלי עם סימן. במהלך ריצת התוכנית יש רגיסטרים אליהם יש להתייחס כאל מספר עם סימן (לעומתם יש רגיסטרים שמייצגים כתובת בזיכרון וכן מייצגים מספר חיובי.) הקלט של הרגיסטרים מיוצג ע"י 3 ספרות הקסה, לכן ע"י 12 ביטים בבינארית, ולכן בשיטת המשלים ל-2 נקבל שהטווח הוא [-2048, 2047]. נמיר לחיובי או שלילי בהתאם ל-MSB.

```
int twos_complement(char* hex) {
    /*make 3 chars hex to 8 chars using sign extention*/
    char* new_hex;
    char sign;
    if (hex[0] <= '7')
    {
        return (int)strtol(hex, NULL, 16);
    }

    new_hex = (char*)malloc(sizeof(char)*9); /* allocate 9 chars for 8 hex digits and the null
terminator */

    new_hex[7] = hex[2];
    new_hex[6] = hex[1];
```

Back to top

```
    new_hex[5] = hex[0];
    /*if the number is negative, make the 3 chars hex to 8 chars using sign extention*/
    for (int i = 0; i < 5; i++){
        new_hex[i] = 'F';
    }
    new_hex[8] = '\0';
    /*new_hex from hex to binary*/
    char* bin = hex_to_bin(new_hex);
    /*flip each bit*/
    for (int i = 0; i < 32; i++) {
        if (bin[i] == '1'){
            bin[i] = '0';
        } else{
            bin[i] = '1';
        }
    }
    /*add one to the binary*/
    for (int i = 31; i >= 0; i--) {
        if (bin[i] == '0') {
            bin[i] = '1';
            break;
        } else {
            bin[i] = '0';
        }
    }
    /*convert the binary to decimal*/
    int decimal = (int)strtol(bin, NULL, 2);
    return -decimal;
}
```

hex_to_bin - ממיר תו הקסה לבינארית. משמש לצורך פונקצית המשלים ל-2.

```
char* hex_to_bin(char* hex) {
    // Lookup table for the binary representation of each hexadecimal digit
    char* lookup_table[16] = {
        "0000", "0001", "0010", "0011",
        "0100", "0101", "0110", "0111",
        "1000", "1001", "1010", "1011",
        "1100", "1101", "1110", "1111"
    };
    // Allocate memory for the binary string
    char* bin = malloc(33); // 32 characters + null terminator
```

Back to top

```
    if (bin == NULL) {
        printf("Failed to allocate memory for bin\n");
        return NULL;
    }
    // Null-terminate the string
    bin[32] = '\0';
    // Convert each character of the hexadecimal string
    for (int i = 0; i < 8; i++) {
        int index;
        if (hex[i] >= '0' && hex[i] <= '9') {
            index = hex[i] - '0';
        } else if (hex[i] >= 'A' && hex[i] <= 'F') {
            index = hex[i] - 'A' + 10;
        } else if (hex[i] >= 'a' && hex[i] <= 'f') {
            index = hex[i] - 'a' + 10;
        } else {
            printf("Invalid hexadecimal digit: %c\n", hex[i]);
            free(bin);
            return NULL;
        }
        strncpy(bin + i*4, lookup_table[index], 4);
    }
    return bin;
}
```

פונקציות אריתמטיות

כל הפונקציות האריתמטיות מקבלות את מערך הרגיסטרים ואינדקסים לרגיסטרים המתאימים. התוצאה מתעדכנת ישירות במערך הרגיסטרים. לבסוף חוזרת ה- PC הבא. כל הפעולות עובדות כפי שמתואר בטבלת ההוראות בעמוד 3. נראה דוגמה לפונקציה אחת ויתר הפונקציות פעולות באותו עיקרון. נשים לב כי נסתכל על המספרים ברגיסטרים על מספרים בעלי סימן ולכן כאשר הם מגיעים לפונקציות אלו הם מספרים בעלי סימן.

add – מחברת שני מספרים על לפי הוראת add שבטבלת ההוראות. ערך הרגיסטר המתאים במערך הרגיסטרים מתעדכן בפונקציה. מוחזר ערך הPC הבא.

```
/*
add - add reg_array[rs], reg_array[rt], reg_array[rm] and store the result in reg_array[rd]
input - *reg_array, rd, rs, rt, rm, current_instruction
*/
int add(int *reg_array, int rd, int rs, int rt, int rm, int current_instruction){
    reg_array[rd] = reg_array[rs] + reg_array[rt] + reg_array[rm];
```

Back to top

```
    return current_instruction + 1;
}
```

פונקציות זיכרון – lw, sw

פועלות לפי טבלת ההוראות בעמוד 3. מקבלות \ מאחסנות ממידע ממערך \למערך הזיכרון. כל הרגיסטרים מתקבלים כמספרים עם סימן ועבור הרגיסטרים שמייצגים כתובת אנחנו מבצעים את ההמרות הנדרשות. נראה דוגמה לאחת מהן.

sw – מאחסנת מידע מהרגיסטר המבוקש לתוך הכתובת המבוקשת בזיכרון. נשים לב כי עבור הרגיסטרים שמייצגים כתובות בזיכרון ביצענו המרה למספר חיובי ולאחר מכן לקחנו את 12 הביטים התחתונים.

```
/*
sw- store data from reg_array[rd] + reg_array[rm] in dataMemory[reg_array[rs] +
reg_array[rt]]
input - *reg_array, *dataMemory, rd, rs, rt, rm
*/
int sw(int *reg_array, char **dataMemory, int rd, int rs, int rt, int rm, int current_instruction,
unsigned int unsigned_imm1, unsigned int unsigned_imm2){
    if (reg_array == NULL || dataMemory == NULL) {
        printf("Error: Null pointer detected\n");
        return current_instruction + 1;
    }
    int old_imm1 = reg_array[1];
    int old_imm2 = reg_array[2];
    reg_array[1] = (int)unsigned_imm1;
    reg_array[2] = (int)unsigned_imm2;
    int place_in_memory = reg_array[rs] + reg_array[rt];

    if (place_in_memory == 0) {
        // Cannot write to memory zero
        return current_instruction + 1;
    }

    if (place_in_memory >= MEMORYSIZE || place_in_memory < 0) {
        // Adjust to fit within memory bounds
        place_in_memory = place_in_memory & 0x0FFF;
    }
    reg_array[1] = old_imm1;
    reg_array[2] = old_imm2;
    char result_str[9];  // Assuming 8 characters for hexadecimal representation + 1 for null
terminator
    snprintf(result_str, sizeof(result_str), "%08X", reg_array[rd] + reg_array[rm]);
```

Back to top

```
    // Allocate memory for the string and copy the result
    char *result_copy = strdup(result_str);
    if (result_copy == NULL) {
        printf("Error: Memory allocation failed\n");
        return current_instruction + 1;
    }

    // Store the data in the memory
    if (dataMemory[place_in_memory] != NULL) {
        free(dataMemory[place_in_memory]);  // Free previously allocated memory
    }
    dataMemory[place_in_memory] = result_copy;
    return current_instruction + 1;
}
```

פונקציות קפיצה

כולל את הפונקציות: beq, bne, blt, bgt, ble, bge, jal. כל הפקודות פועלות לפי טבלת ההוראות בעמוד 3. כל הפונקציות פועלות בצורה דומה בכך שהן בודקות אם מתקיים תנאי כלשהו (פרט ל-jal) ואם כן מחזירות את הכתובת שבrm ובכך הן מעדכנות את pc להיות הכתובת שב-rm שכן אנו מתייחסים לערך ההחזרה כאל ה-pc הבא. אם לא, הן מחזירות pc +1 כלומר לא ביצענו קפיצה. נעדכן את הרגיטסרים שמייצגים כתובות למספרים חסרי סימן וניקח את 12 הביטים התחתונים. נראה דוגמה לפונקציה אחת.

beq – אם ערכי הרגיטסרים המתאימים שווים מחזירה את הכתובת ב-rm ובכך מעדכנת את pc להיות מה שיש בכתובת זו. אחרת מקדמת את pc ב-1.

```
/*
beq - if reg_array[rs] == reg_array[rt] then jump to the instruction in the address
reg_array[rm][low bits- 11:0]
input - *reg_array, rd, rs, rt, rm, current_instruction
*/
int beq(int *reg_array, int rd, int rs, int rt, int rm, int current_instruction, unsigned int
unsigned_imm1, unsigned int unsigned_imm2){

    if (reg_array[rs] == reg_array[rt])
    {
        reg_array[1] = (int)unsigned_imm1;
        reg_array[2] = (int)unsigned_imm2;
        /*if the values are equal, jump to the address in rm*/
        return (reg_array[rm] & (0x0FFF));
    }
```

Back to top

```
    return current_instruction + 1; /*if the values are not equal, continue to the next instruction*/
}
```

פונקציות IO

הפונקציות – reti, in, out. נפרט על כל אחת מהן.

reti - אחראית לניהול תהליך החזרה מ-interrupt. היא מאפסת את הדגל של ה-irq, בודקת האם אנחנו בתוך טיפול ב-irq, אם כן מחזירה את כתובת ה-handler, אחרת מחזירה את הכתובת שברגיסטר המתאים (irqreturn).

```
 /*
 reti - return from interrupt, put the value in IOregisterd[rd] in the register rd
 input - *IOregisterd, *reg_array, rd, rs, rt, rm, current_instruction
  */
int reti(int *IOregisterd, int inside_irq, int current_instruction){
    int temp_ins;
    int called_from_reti;
    inside_irq = 0; /*we are not inside an irq*/

    called_from_reti = 1;
    temp_ins = irq_handler(IOregisterd, inside_irq, current_instruction, called_from_reti);
    if (temp_ins != -1)
    {
        /*if we are inside an irq, return the address of the handler*/
        return temp_ins;
    }
    /*if we are not inside an irq, return the address in irqreturn*/
    return IOregisterd[7];
}
```

in – מכניסה את המידע שיושב בדיסק בכתובת שנמצאת ברגיסטר ה-IO המתאים לתוך רגיסטר היעד. לבסוף קוראת לפונקציה שמדפיסה לhwregtrace. נשים לב כי בהתאם לתוכן הרגיסטר המתאים מתבצעות ההמרות הנדרשות. למשל, אם הוא מייצג כתובת אז נאפשר רק מספרים חיוביים.

```
/*
in - put the value in io_array[reg_array[rs] + reg_array[rt]] in reg_array[rd]
input - rd, rs, rt, *reg_array, *io_array, *hwregtrace
*/
void in(int rd, int rs, int rt, int *reg_array, int *io_array, int* current_cycle, FILE* hwregtrace) {
    int reg_num = reg_array[rs] + reg_array[rt];

    if (reg_num == 22) {
```

Back to top

```
        reg_array[rd] = 0; // A read from monitorcmd using the in instruction will return the value 0
    }

    else {
        if(reg_num == 6 | reg_num == 7 | reg_num == 16 ) {//registers representing addresses
            reg_array[1] = (int)io_array[1];
            reg_array[2] = (int)io_array[2];
            reg_array[rd] = io_array[reg_num] &0xFFF;

        } else if (reg_num == 20) {
            reg_array[1] = (int)io_array[1];
            reg_array[2] = (int)io_array[2];
            reg_array[rd] = io_array[reg_num] &0xFFFF;

        } else {
            reg_array[rd] = io_array[reg_num]; // Updating $rd register with the IO input
        }

    }
    hwregout_printer(reg_num, 0, io_array, current_cycle, hwregtrace);
}
```

out – מוציאה את המידע שיושב ברגיסטר המתאים ומכניסה אותו לתוך הכתובת בדיסק ששמורה ברגיסטר המתאים. מדפיסה עדכון לקובץ המתאים במידת הצורך (לקובץ leds או לקובץ display7seg) ומעדכנת את רגיסטר המוניטור במידת הצורך. לבסוף קוראת לפונקציה שמדפיסה לhwregtrace. בדומה ל-in, מתבצעות ההמרות לרגיסטרים הנדרשים.

```
/*
out - put the value in reg_array[rm] in io_array[reg_array[rs] + reg_array[rt]]
input - rm, rs, rt, *reg_array, *io_array, *hwregtrace_filename, *leds_filename, display7seg_filename, screen
*/
void out(int rm, int rs, int rt, int *reg_array, int *io_array, int* current_cycle, char** memlines, char** disklines, FILE *hwregtrace, FILE *leds, FILE *display7seg, int screen[SCREENSIZE], unsigned int unsigned_imm1, unsigned int unsigned_imm2) {
    int reg_num = reg_array[rs] + reg_array[rt];

    if(reg_num == 6 | reg_num == 7 | reg_num == 16 ) {//registers representing addresses
        reg_array[1] = (int)unsigned_imm1;
        reg_array[2] = (int)unsigned_imm2;
```

Back to top

```
        io_array[reg_num] = reg_array[rm] &0xFFF; // Updating the IO register with the value in
$rm
    }

    else if (reg_num == 20){
        reg_array[1] = (int)unsigned_imm1;
        reg_array[2] = (int)unsigned_imm2;
        io_array[reg_num] = reg_array[rm] &0xFFFF; // 16 bits for monitoraddr
    }

    else{
        io_array[reg_num] = reg_array[rm]; // Updating the IO register with the value in $rm
    }

    if (reg_num == 9) {
        //If leds are been changed, the clock cycle and the status of all leds are written to
leds.txt
        fprintf(leds, "%d %08X\n", *current_cycle, io_array[9]);
    }
    if (io_array[22] == 1) { // monitorcmd register
        screen[io_array[20]] = io_array[21];
    }
    if(reg_num == 10) {
        fprintf(display7seg, "%d %08X\n", *current_cycle, io_array[10]);
    }
    if(reg_num == 14) {
        irq1_check(io_array, disklines, memlines);
    }
    hwregout_printer(reg_num, 1, io_array, current_cycle, hwregtrace);

    if(io_array[22] == 1) {
        io_array[22] = 0; // Reset monitor command after update;
    }
}
```

פונקציות הדפסה לקבצים

הפונקציות – print_memory_array_to_file, write_to_regout, monitoryuv_printer, write_to_trace, hwregout_printer. פונקציות אלו מדפיסות אל קבצי הפלט של התוכנית. חלקן מדפיסות במהלך הריצה שכן הן נדרשות לעדכן את הקובץ לאחר ביצוע פעולה כלשהי, וחלק נקראות רק בסיום התוכנית.

Back to top

נפרט עבור כל קובץ פלט מה מופיע בו ומאיזו פונקציה הוא מתקבל:

*כל הקבצים הם מסוג txt פרט לקובץ monitor.yuv.

dmemout – מתקבל מ- print_memory_array_to_file. מכיל את תוכן הזיכרון בסיום הריצה.
diskout – מתקבל מ- print_memory_array_to_file. מכיל את תוכן הדיסק בסיום הריצה.
regout – מתקבל מ- write_to_regout. מכיל את ערכי הרגיסטרים R3-R15 בסיום הריצה.
trace – מתקבל מ- write_to_trace. מכיל שורה עבור כל הוראה שבוצעה ע"י המעבד שמורכבת מ-PC, הפקודה, וערכי הרגיסטרים R3-R15 לפני הרצת הפקודה.
hwregtrace – מתקבל מ- hwregout_printer. מכיל שורה עבור כל הוראת קריאה או כתיבה מרגיסטר IO. שורה זו כוללת את ה-cycle בו בוצעה, אם זו פקודה קריאה או כתיבה, שם הרגיסטר והמידע שנקרא או נכתב.
cycles – מתקבל ע"י הרצת פקודת הדפסה פשוטה שמדפיסה את ה-cycle בסוף הריצה.
leds – מתקבל ע"י הרצת פקודת הדפסה פשוטה שמדפיסה לקובץ כאשר בוצעה פעולה כלשהי על רגיסטר IO מספר 9 שמשמש ללדים. פקודה זו תרוץ במידת הצורך בפונקציית IO.
monitor – מתקבל מ - monitoryuv_printer. מכיל את ערכי הפיקסלים של המסך בסוף הריצה.
monitor.yuv – מתקבל מ - monitoryuv_printer. מכיל את ערכי הפיקסלים של המסך בסוף הריצה בקובץ בינארי שניתן להציגו גרפית.

פונקציות לטיפול בפסיקות

ישנן שלוש פסיקות בהן תומך המעבד: irq0, irq1, irq2.
irq0 – פסיקה 0 שייכת לטיימר. ניתן להגדיר בקוד האסמבלי את ערך הטיימר המקסימלי.
Irq1 – פסיקה 0 שייכת לדיסק. בעזרתה הדיסק מודיע למעבד שהוא סיים לבצע פקודת קריאה או כתיבה. כל פקודה כזו לוקחת 1024 מחזורי שעון.
irq2 – פסיקה חיצונית. בקובץ הקלט irq2in.txt מופיעים מחזורי השעון בהם הפסיקה מופיעה.

לצורך הטיפול בפסיקות הגדרנו שלוש פונקציות בודקות עבור כל אחת מהפסיקות ופונקציית handler.
בתחילת כל פונקציה בודקת אנחנו מאפסים את רגיסטר הסטטוס המתאים, כדי שאם תתקבל פסיקה נוספת במהלך הרצת השגרה נדע על כך בסיום הרצתה. נשים לב שרגיסטר המונה המתאים לכל פסיקה, שבודק כמה מחזורי שעון עברו, אכן מתקדם עבור כל פונקציה בודקת בכל מחזור שעון שכן יש קריאה לכל פונקציה כזו בכל מחזור שעון.

irq0_checker- בתחילה מתבצע איפוס רגיסטר הסטטוס המתאים ואז בדיקה אם הפסיקה מאופשרת. אם כן מקדמים את המונה של השעון באחד. אם הגענו למספר שהוגדר מקסימלי, אז נשנה את הסטטוס ל-1 ונאפס את הטיימר.

```
//irq0_handler
/*In every clock cycle in which the timer is enable, the timercurrent register is incremented by one
and implementing series of checks.*/
```

Back to top

```
void irq0_checker(int* io_array){
    io_array[3] = 0;  /* irq0status is off */
    if (io_array[11] == 1) {// Check if timerenable register is on
        io_array[12] += 1; // Incrementing timercurrent register by one
        if (io_array[12] == io_array[13]){ // check if timercurrent == timermax
            io_array[3] = 1; // irq0status is on
            io_array[12] = 0; //reset timercurrent
        }
    }
}
```

irq1_check ו- disk_timer – משמשים לצורך ניהול הטיפול ב-interrupt1 הקשור לדיסק.
irq1_check - בתחילה נבדוק האם הדיסק לא פנוי. אם כן, נצא מהפונקציה ומחוצה לה נקרא ל-disk_timer.
אחרת, הדיסק פנוי, לכן נבצע פעולת קריאה או כתיבה בהתאם לתוכן הרגיסטר המתאים. לבסוף נעדכן את הרגיסטרים הנחוצים.
disk_timer - בתחילה מתבצע איפוס רגיסטר הסטטוס המתאים ואז בדיקה אם הדיסק עסוק. אם כן, נקדם את המונה ב-1. אם המונה גדול מ-1024 (מספר מחזורי השעון שלוקח לבצע פקודת קריאה או כתיבה) אז נהפוך את הרגיסטר המתאים לפנוי ונדליק את הסטטוס.

```
// irq1_check() checks if disk is busy or free and performs the read/write operation
// io_array: array of IO registers
// disk: array of disk contents
// memin: array of memory contents
void irq1_check(int* io_array, char** disk, char** memin){
    if (io_array[17] != 0) return; // check if disk is busy
    int sectorfirstrow = io_array[15]*128 ; // get the sector number

    if (io_array[14] == 1){ //reading from disk
        for (int i = 0; i < 128; i++) {// read 128 words from disk
            strcpy(memin[(i + io_array[16]) % MEMORYSIZE], disk[sectorfirstrow + i]);
            }
        }

    if(io_array[14] == 2){ //writing to disk
        for (int i = 0; i < 128; i++) {// write 128 words to disk
            strcpy(disk[sectorfirstrow + i], memin[(i + io_array[16]) % MEMORYSIZE]);
            }

        }
```

Back to top

```
    io_array[17] = 1; // disk is busy
    diskcycle = 0; // reset diskcycle
}
```

```
/*update the disk timer if the disk is busy, if diskCycle reached 1024 or above, init the
relevent registers*/
void disk_timer(int* io_array) {
    io_array[4] = 0; // irq1status is off
    if (io_array[17]) {
        diskcycle++; // if disk is busy, increment diskcycle
        // reached 1024 cycles
        if (diskcycle >= 1024) {
            // reset disk command
            io_array[14] = 0;
            // and disk status
            io_array[17] = 0;
            // and trips irq1status
            io_array[4] = 1;
        }
    }
}
```

irq2_checker- בתחילה מתבצע איפוס רגיסטר הסטטוס המתאים ואז בדיקה האם אנו בפסיקה החיצונית הראשונה, אם כן נקרא את מחזור השעון הראשון מהקובץ. בהמשך, אם עברנו את מחזור השעון של הפסיקה נקדם את מחזור השעון של הפסיקה לפסיקה הבאה. אם הגענו למחזור השעון של הפסיקה ונדליק את הסטטוס.

```
/*
external (to the processor) interrupt line, An interrupt file to the simmulator specifies when
the interrupt occurs
input - *IO_array: io registers array, *irq2in: file with the interrupt, current_interrupt - current
interrupt, init with -1 outside the function
int current_interrupt - current interrupt, init with -1 outside the function
ouput - None(void)
*/
void irq2_checker(int* IO_array, FILE* irq2in, int current_interrupt) {
    IO_array[5] = 0; /* irq2status is off */
    if (current_interrupt  == -1)
    {
        /*first time we enterted the function*/
```

Back to top

```
        current_interrupt = fscanf(irq2in, "%d", &current_interrupt);
    }

    /*if the current cycle is greater than the current interrupt (we passed it) and the file is not
empty, read the next interrupt*/
    if (current_cycle > current_interrupt && !feof(irq2in))
    {
        current_interrupt = fscanf(irq2in, "%d", &current_interrupt);
        /*fscand didn't Succeeded*/
        if (current_interrupt != 1)
        {
            printf("Error reading file");
            exit(1);
        }
    }
    /* the current interrupt is the current cycle, put the value in the IO array */
    if (current_interrupt == current_cycle)
    {
        /*if the interrupt is enabled, turn on irq2status*/
        IO_array[5] = 1;
    }
    if (!feof(irq2in))
    {
        fclose(irq2in);
    }
}
```

irq_handler- מבצעת את הבדיקה:

irq = (irq0enable & irq0status) | (irq1enable & irq1status) | (irq2enable & irq2status)

אם נקבל 1 אז נבדוק קודם אם אנחנו באמצע הרצת שגרה נצא מהפונקציה. אחרת, נאפס את הביטים של הסטטוס לשלושת הרגיסטרים, נדליק את הדגל שאומר שאנחנו בתוך פסיקה, נשמור את ה-PC כדי שנוכל לדעת לאן לחזור, ומחזיר את הכתובת של השגרה.

```
/*
interups handler
input - *IO_array: io registers array, inside_irq - if we are inside an irq, current_instruction -
the current instruction
output - return current instruction if we are inside an irq (we don't support nested irq), the
address of the handler if we are not
 and -1 if none of the enables and their corresponding states are on
*/
```

Back to top

```
int irq_handler(int* IO_array, int inside_irq, int current_instruction) {
   if((IO_array[0] ==  1 && IO_array[3] == 1) || (IO_array[1] == 1 && IO_array[4] == 1) ||
(IO_array[2] == 1 && IO_array[5] == 1)) {
      /*interruption had happen if and only if it at least one enable and its corresponding
status are on*/
      if(inside_irq){
         /*if we are already inside an irq, countinue (ignore nested irq)*/
         return current_instruction;
      }
      IO_array[3] = 0; /* irq0status is off */
      IO_array[4] = 0; /* irq1status is off */
      IO_array[5] = 0; /* irq2status is off */
      inside_irq = 1; /*we are inside an irq*/
      IO_array[7] = current_instruction; /* save the current address in irqreturn so we can
return to it later on*/
      return IO_array[6]; /*return the address of the handler*/
   }
   return -1;
}
```

פונקציות להרצת הקוד והרצת הפקודות ופונקציית ה-main

יש שלוש פונקציות עיקריות שמריצות את הקוד:

1. ecxecute_instruction- מיועדת להרצת הפקודות שבקובץ imemin.
2. run_code - מריצה מספר פונקציות עיקריות, בפרט את ecxecute_instruction.
3. main – מקבלת ארגומנטים מהמשתמש (הקבצים) ומריצה מספר פונקציות, הפרט את run_code.

נפרט על כל אחת מהפונקציות:

ecxecute_instruction – פונקציה זו מריצה פקודה בודדת. היא מקבלת מערך ובו כל ההוראות. בשלב הראשון היא יוצרת מערך מתאים עבור פקודה אחת שבו יש את הרגיסטרים וה-opcode. לאחר מכן היא משתמשת בבלוק switch-case להרצת ה-opcode המתאים.
לאחר מכן, הפונקציה מקדמת את מונה ההוראות ובמידה שהוא גדול ממספר ההוראות הכולל יוצאת עם exit. לבסוף, מקדמת את מחזור השעון – משתנה שנועד להחזיר את מספר מחזורי השעון שלקח לתוכנית לרוץ, ואת הרגיסטר המתאים של מחזור השעון – רגיסטר IO שצריך להיות מעודכן. כמו כן, אם תוכן הרגיסטר של מחזור השעון מגיע למספר המקסימלי הוא מתאפס. לבסוף כותבת ל-trace המתאים.

```
/*
ecxecute the instruction in the code
inputs- char** instructions,char** dataMemoryLines, int* registers, int* IOregisterd, int*
dataMemory, int current_instruction, char* trace_filename, FILE* hwregtrace, FILE* leds,
FILE* display7seg, int* screen
```

Back to top

```
output- 1 if the program should exit, 0 otherwise
*/
int ecxecute_instruction(char** instructions,char** dataMemoryLines, char** diskLines, int*
registers, int* IOregisterd, int instructions_size, int* current_instruction, int*
current_cycle,FILE* trace, FILE* hwregtrace, FILE* leds, FILE* display7seg, int* screen,
size_t* line_amount)
{
    char* instruction;
    int opcode;
    char opcode_char[3];
    int rd;
    char rd_char[2];
    int rs;
    char rs_char[2];
    int rt;
    char rt_char[2];
    int rm;
    char rm_char[2];
    int imm1;
    unsigned int unsigned_imm1;
    char imm1_char[4];
    int imm2;
    unsigned int unsigned_imm2;
    char imm2_char[4];
    int i;
    int dataMemoryLines_size;
    int value;
    int next_instruction;
    int exit;
    int temp_ins;
    char newline[2] = "\n";
    int keep_rd;
    temp_ins = -1;
    exit = 0;

    instruction = instructions[(*current_instruction)];
    /*copy opcode*/
    strncpy(opcode_char, instruction, 2); opcode_char[2] = '\0';
    opcode = (int)strtol(opcode_char, NULL, 16);
    /*copy rd*/
    strncpy(rd_char, &instruction[2], 1); rd_char[1] = '\0';
```

Back to top

```
rd = (int)strtol(rd_char, NULL, 16);
/*copy rs*/
strncpy(rs_char, &instruction[3], 1); rs_char[1] = '\0';
rs = (int)strtol(rs_char, NULL, 16);
/*copy rt*/
strncpy(rt_char, &instruction[4], 1); rt_char[1] = '\0';
rt = (int)strtol(rt_char, NULL, 16);
/*copy rm*/
strncpy(rm_char, &instruction[5], 1); rm_char[1] = '\0';
rm = (int)strtol(rm_char, NULL, 16);
/*copy imm1*/
strncpy(imm1_char, &instruction[6], 3); imm1_char[3] = '\0';
unsigned_imm1 = strtoul(imm1_char, NULL, 16);
/*copy imm2*/
strncpy(imm2_char, &instruction[9], 3); imm2_char[3] = '\0';
unsigned_imm2 = strtoul(imm2_char, NULL, 16);
imm1 = twos_complement(imm1_char);
imm2 = twos_complement(imm2_char);
registers[1]=imm1;
registers[2]=imm2;

keep_rd = registers[rd];
write_to_trace(registers, *current_instruction, instruction, trace);
switch (opcode)
{
case 0:
    /*add*/
    next_instruction = add(registers, rd, rs, rt, rm, (*current_instruction));
    break;
case 1:
    /*sub*/
    next_instruction = sub(registers, rd, rs, rt, rm, (*current_instruction));
    break;
case 2:
    /*mac*/
    next_instruction = mac(registers, rd, rs, rt, rm, (*current_instruction));
    break;
case 3:
    /*and*/
    next_instruction = and(registers, rd, rs, rt, rm, (*current_instruction));
    break;
```

Back to top

```
    case 4:
        /*or*/
        next_instruction = or(registers, rd, rs, rt, rm, (*current_instruction));
        break;
    case 5:
        /*xor*/
        next_instruction = xor(registers, rd, rs, rt, rm, (*current_instruction));
        break;
    case 6:
        /*sll*/
        next_instruction = sll(registers, rd, rs, rt, rm, (*current_instruction));
        break;
    case 7:
        /*sra*/
        next_instruction = sra(registers, rd, rs, rt, rm, (*current_instruction));
        break;
    case 8:
        /*srl*/
        next_instruction = srl(registers, rd, rs, rt, rm, (*current_instruction));
        break;
    case 9:
        /*beq*/
        next_instruction = beq(registers, rd, rs, rt, rm, (*current_instruction), unsigned_imm1,
unsigned_imm2);
        break;
    case 10:
        /*bne*/
        next_instruction = bne(registers, rd, rs, rt, rm, (*current_instruction), unsigned_imm1,
unsigned_imm2);
        break;
    case 11:
        /*blt*/
        next_instruction = blt(registers, rd, rs, rt, rm, (*current_instruction), unsigned_imm1,
unsigned_imm2);
        break;
    case 12:
        /*bgt*/
        next_instruction = bgt(registers, rd, rs, rt, rm, (*current_instruction), unsigned_imm1,
unsigned_imm2);
        break;
    case 13:
```

Back to top

```
        /*ble*/
        next_instruction = ble(registers, rd, rs, rt, rm, (*current_instruction), unsigned_imm1,
unsigned_imm2);
        break;
    case 14:
        /*bge*/
        next_instruction = bge(registers, rd, rs, rt, rm, (*current_instruction), unsigned_imm1,
unsigned_imm2);
        break;
    case 15:
        /*jal*/
        next_instruction = jal(registers, rd, rs, rt, rm, (*current_instruction), unsigned_imm1,
unsigned_imm2);
        break;
    case 16:
        /*lw*/
        next_instruction = lw(registers, dataMemoryLines, rd, rs, rt, rm, (*current_instruction),
unsigned_imm1, unsigned_imm2);
        break;
    case 17:
        /*sw*/
        next_instruction = sw(registers, dataMemoryLines, rd, rs, rt, rm, (*current_instruction),
unsigned_imm1, unsigned_imm2);
        break;
    case 18:
        /*reti*/
        next_instruction = reti(IOregisterd, inside_irq, (*current_instruction));
        break;
    case 19:
        /*in*/
        in(rd, rs, rt, registers, IOregisterd, current_cycle, hwregtrace);
        next_instruction = (*current_instruction) + 1;
        break;
    case 20:
        /*out*/
        out(rm, rs, rt, registers, IOregisterd, current_cycle, dataMemoryLines, diskLines,
hwregtrace, leds, display7seg, screen, unsigned_imm1, unsigned_imm2);
        next_instruction = (*current_instruction) + 1;
        break;
    default: /*case 21- halt*/
        exit = 1; /*exit the loop*/
```

Back to top

```
        break;
    }
    if(next_instruction >= instructions_size){
        /*if the next instruction is out of bounds, exit the loop*/
        exit = 1;
    }
    if(rd ==0 || rd == 1 || rd == 2){
        /*if this the register is 0, 1, or 2, don't write to it*/
        registers[rd] = keep_rd;
    }
    /*print current instruction in trace file*/
    (*current_cycle)++; /*increment the cycle*/
    IOregisterd[8] = IOregisterd[8] + 1; /*increment the clock cycle*/
    if (IOregisterd[8]  == UINT32_MAX) /*if the clock cycle is at its maximum value, reset it*/
    {
        IOregisterd[8] = 0 ; /*reset the clock cycle*/
    }

    (*current_instruction) = next_instruction;
    return exit; /*return 1 if the program should exit, 0 otherwise*/

}
```

run_code – תחילה, הפונקציה פותחת מספר קבצים לקריאה ולכתיבה. החלק העיקרי בפונקציה הוא לולאת while שכל עוד קיבלנו ערך 0 בחזרה מהפונקציה ecxecute_instruction נמשיך להריץ את הפקודות, וברגע שנקבל 1 נסיים את הרצתן. לאחר מכן הפונקציה מדפיסה לקבצים אותם יש להדפיס פעם אחת בסיום הריצה ולבסוף סוגרת אותם.

```
/*
run the functions in the code
input- char** instructions,char** dataMemoryLines, char** disk_lines, char* dmemout, char*
diskout, char* regout, char* trace_filename,
char* irq2in, char* hwregtrace_filename, char* leds_filename, char* display7seg_filename,
char* monitortxt, char* monitoryuv, char* cycles
output- None(void)
*/
void run_code(char** instructions,char** dataMemoryLines, char** disk_lines, const char*
dmemout, const char* diskout, const char* regout, const char* trace_filename, const char*
forirq2in, const char* hwregtrace_filename, const char* leds_filename,
 const char* display7seg_filename, const char* monitortxt, const char* monitoryuv, const
char* cycles_filename, size_t* line_amount){
```

Back to top

```
int i;
int instructions_size;
int registers[REGISTERS_COUNT] = {0};
int IOregisterd[IOREGISTERS_COUNT] = {0};
int screen[SCREENSIZE];
for(i = 0; i < SCREENSIZE; i++) {
    screen[i] = 0;
}
int value;
int exit_prog;
int current_interrupt_irq2 = -1; /*init with -1*/
int called_from_reti;
int current_instruction = 0; /*index of the current instruction start with the first in place 0*/
int current_cycle = 0; /*start with cycle 0*/
exit_prog = 0; /*exit flag*/
inside_irq = 0; /*we are not inside an irq*/

FILE *irq2in = fopen(forirq2in, "r");
FILE *trace = fopen(trace_filename, "w");
FILE *hwregtrace = fopen(hwregtrace_filename, "w");
FILE *leds = fopen(leds_filename, "w");
FILE *display7seg = fopen(display7seg_filename, "w");
FILE *cycles = fopen(cycles_filename, "w");
int loop = 0;

if(irq2in == NULL){
    printf("Error opening file");
    exit(1);
}

/*start irq_2*/
irq2_checker(IOregisterd, irq2in ,&current_interrupt_irq2, &current_cycle);

instructions_size = *line_amount;

while(exit_prog == 0){
    /*run the code*/
    /*interups*/
    disk_timer(IOregisterd); /*disk timer*/
    irq0_checker(IOregisterd);
    irq2_checker(IOregisterd, irq2in, &current_interrupt_irq2, &current_cycle);
```

Back to top

```
        called_from_reti = 0;
        current_instruction = irq_handler(IOregisterd, 0, current_instruction, called_from_reti);
        exit_prog = ecxecute_instruction(instructions, dataMemoryLines, disk_lines, registers,
IOregisterd,
        instructions_size, &current_instruction, &current_cycle,trace, hwregtrace, leds,
display7seg, screen, line_amount);

        if (loop == 100)
        {
            //break;

        }
        loop++;

    }

    /*write the data memory to the file*/
    print_memory_array_to_file(dataMemoryLines, dmemout, MEMORYSIZE);
    /*write the disk to the file*/
    print_memory_array_to_file(disk_lines, diskout, DISKSIZE);
    /*write the registers to the file*/
    write_to_regout(registers, regout);
    /*write the screen to the file*/
    monitoryuv_printer(screen, monitortxt, monitoryuv);
    /*write the cycles to the file*/
    fprintf(cycles, "%d\n", current_cycle);

    /*close the files*/
    fclose(irq2in);
    fclose(trace);
    fclose(hwregtrace);
    fclose(leds);
    fclose(display7seg);
    fclose(cycles);
    return;
}
```

main– בפונקציה זו אנו מקבלים את הארגומנטים המתקבלים מהרצת שורת הפקודה. לאחר מכן פותחים את קבצי הקלט imemin וdmemin וקוראים לפונקציה שמכניסה את המידע מהקובץ למערך מתאים. לאחר מכן אנו קוראים לפונקציה run_code.

Back to top

```c
int main(int argc, char *argv[]) {
    const char *forimemin = argv[1];
    const char *dmemin = argv[2];
    const char *diskin = argv[3];
    const char *irq2in = argv[4];
    const char *dmemout = argv[5];
    const char *regout = argv[6];
    const char *trace = argv[7];
    const char *hwregtrace = argv[8];
    const char *cycles = argv[9];
    const char *leds = argv[10];
    const char *display7seg = argv[11];
    const char *diskout = argv[12];
    const char *monitortxt = argv[13];
    const char *monitoryuv = argv[14];
    size_t line_amount;
    if (argc != 15) {
        printf("Error in ARGS");
        return 1;
    }
    /*read imemin into array*/
    FILE *imemin = fopen(forimemin, "r");
    if(imemin == NULL){
        printf("Error opening file");
    }
    char **imemin_lines = read_file_to_array(IMEMIN_LINE_LENGTH, imemin,
&line_amount);
    fclose(imemin);

    /*read imemin into array*/
    char **dmemin_lines = from_file_to_array(dmemin, MEMORYSIZE,
DMEMIN_LINE_LENGTH);
    /*read dmemin into array*/
    char **disk_lines = from_file_to_array(diskin, DISKSIZE, DISK_LINE_LENGTH);

    run_code(imemin_lines,dmemin_lines, disk_lines, dmemout, diskout, regout, trace, irq2in,
hwregtrace, leds, display7seg, monitortxt, monitoryuv, cycles, &line_amount);
    return 0;
}
```

Back to top

תוכניות אסמבלי לבדיקה

ישנן ארבע פונקציות אסמבלי שנועדו לבדיקת תקינות האסמבלר והסימולטור. נסביר מה כל אחת מהן עושה.

mulmat.asm

מבצעת הכפלה של שתי מטריצות ריבועיות מסדר 4. הפונקציה מורכבת מ-main שם מתבצעת הקצאת מקום במחסנית, אחסון ערך ההחזרה ואתחול רגיסטר. החלק העיקרי בקוד הוא לולאות ה-for – יש שלוש לולאות for כאשר הלולאה הראשונה משמשת למעבר על שורות מטריצת התוצאה, הלולאה השנייה על עמודות מטריצת התוצאה, ואז מתבצע אתחול לערך אפס לתאי המטריצה. בתוך הלולאה השלישית מתבצעת ההכפלה עצמה וצבירת התוצאה. לאחר מכן מתבצעת הכיתבה לזיכרון. לבסוף מתבצעת יציאה מהפונקציה וחזרה לפונקציה הקוראת.

```
#Matrix multiplication mat1*mat2 = result_mat
main:
   sll $sp, $imm1, $imm2, $zero, 1, 11 # set $sp = 1 << 11 = 2048
   add $sp, $sp, $imm1, $zero, -3, 0 # allocate space in stack

   sw $s0, $sp, $imm1, $zero, 1, 0 #save s0
   add $s1, $sp, $imm1, $zero, 2, 0 #save s1
   add $s2, $sp, $imm1, $zero, 3, 0 #save s2

   add $s0, $zero, $imm1, $zero, -1, 0 # index i = -1

for1:
   add $s0, $s0, $imm1, $zero, 1, 0 #i++
   beq $zero, $imm2, $s0, $imm1, exit, 4 #if(4 =  i) exit for1
   add $s1, $zero, $imm1, $zero, -1, 0 # index j = -1

for2:

   add $s1, $s1, $imm1, $zero, 1, 0 #j++
   beq $zero, $imm2, $s1, $imm1, for1, 4 #if(4 =  j) exit for2
   add $t0, $zero, $zero, $zero, 0, 0 # temp_sum = 0
   add $s2, $zero, $zero, $zero, 0, 0 # index k = 0

for3:
   beq $zero, $imm2, $s2, $imm1, cal, 4 #if(4 =  k) exit for3 to cal

   mac $t1, $s0, $imm1, $zero, 4, 0 # t1 = i * 4
   add $t1, $imm1, $s2, $t1, 0x100, 0 #mat1[i][k]-> MEM[0x100 + i * 4 + k]
```

Back to top

```
    lw $t1, $t1, $zero, $zero, 0, 0 # t1 <= mat1[i][k]

    mac $t2, $s2, $imm1, $zero, 4, 0 # t2 = k * 4
    add $t2, $imm1, $t2, $s1, 0x110, 0 #mat1[k][j]-> MEM[0x110 + k * 4 + j]
    lw $t2, $t2, $zero, $zero, 0, 0 # t2 <= mat2[k][j]

    mac $t0, $t1, $t2, $t0, 0, 0 # temp sum = mat1[i][k] * mat2[k][j] + temp_sum
    add $s2, $s2, $imm1, $zero, 1, 0 # k = k + 1
    beq $zero, $zero, $zero, $imm1, for3, 0 # jump back to for3

cal:
    mac $t2, $s0, $imm1, $zero, 4, 0 # t2 = i * 4
    add $t1, $imm1, $t2, $s1, 0x120, 0 # result_mat[i][j]-> MEM[0x120 + i * 4 + j]
    sw $t0, $t1, $zero, $zero, 0, 0 # result_mat[i][j] = temp_sum
    beq $zero, $zero, $zero, $imm1, for2, 0 #go to for2

exit:
    lw $s0, $sp, $imm1, $zero, 0, 0 # restore value of $s0
    lw $s0, $sp, $imm1, $zero, 1, 0 # restore value of $s1
    lw $s0, $sp, $imm1, $zero, 2, 0 # restore value of $s2
    add $sp, $sp, $imm1, $zero, 3, 0 # Restore stack
    halt $zero, $zero, $zero, $zero, 0, 0                # Halt the system
```

binom.asm

מחשבת את מקדמי הבינום של ניוטון בצורה רקורסיבית לפי קוד הסי הבא:

```
int binom(n, k)
{
        if (k == 0 || n == k)
        return 1;
        return binom(n-1, k-1) + binom(n-1, k)
}
```

מורכבת מארבעה חלקים עיקריים:

1. main - טעינת משתנים מהזיכרון, קריאה לפונקציה ושמירת ערך החזרה בכתובת המתאימה בזיכרון.
2. binom- מקצה מקום במחסנית הרקורסיה ומאחסנת שם את המשתנים הרלוונטיים. לאחר מכן, בודקת אם אחד מממקרי הקצה מתקיים. אם לא מבצעת שתי קריאות רקורסיביות בהתאם לקוד הסי.
3. base_case - מחזירה 1 וקופצת ל-lable הסיום.
4. end_binom – משחררת את המחסנית וקופצת לכתובת החזרה.

```
    sll $sp, $imm1, $imm2, $zero, 1, 11              #     set $sp = 1 << 11 = 2048
    add $sp, $sp, $imm1, $zero, -2, 0          # Allocate space on the stack
    sw $a0, $sp, $imm1, $zero,  0, 0           # Store Current a0
```

Back to top

```
    sw $a1, $sp, $imm1, $zero,  1, 0            # Store Current a1
    lw $a0, $imm1, $zero, $zero, 0x100, 0       # Load n into $a0 from address
    lw $a1, $imm1, $zero, $zero, 0x101, 0       # Load k into $a1 from address
    jal $ra, $zero, $zero, $imm1, binom, 0      # function call binom, save return addr in $ra
    lw $a0, $sp, $imm1, $zero, 0, 0             # Restore  $a0
    lw $a1, $sp, $imm1, $zero, 1, 0             # Restore $a1
    add $sp, $sp, $imm1, $zero, 2, 0            # Free stack space
    sw $zero, $imm1, $zero, $v0, 0x102, 0       # Store result at address 0x102
    halt $zero, $zero, $zero, $zero, 0, 0       #     halt (exit)

binom:
    add $sp, $sp, $imm1, $zero, -4, 0           # Allocate space on the stack for 4
    sw $a0, $sp, $imm1, $zero, 0, 0             # Store Current n
    sw $a1, $sp, $imm1, $zero, 1, 0             # Store Current k
    sw $s0, $sp, $imm1, $zero, 2, 0             # Store Current $s0
    sw $ra, $sp, $imm1, $zero, 3, 0             # Store Current $ra


    beq $zero, $a1, $zero, $imm1, base_case, 0   # Base cases: binom(n, 0)
    beq $zero, $a0, $a1, $imm1, base_case, 0     # Base cases: binom(n, n)
    blt $zero, $a0, $a1, $imm1,  smaller_case, 0 # Base cases: binom(n, k), n<k

 #    Recursive calls: binom(n-1, k-1) + binom(n-1, k)
    add $a0, $a0, $imm1, $zero, -1, 0           # claculate n-1
    add $a1, $a1, $imm1, $zero, -1, 0           # calculate k-1
    jal $ra, $zero, $zero, $imm1, binom, 0      # Recursive call for binom(n-1, k-1)
    add $s0, $v0, $zero, $zero, 0, 0            # $s0 = binom(n-1, k-1)

    add $a1, $a1, $imm1, $zero, 1, 0            # calculate (k-1)+1
    jal $ra, $zero, $zero, $imm1, binom, 0      # Recursive call for binom(n-1, k)
    add $v0, $v0, $s0, $zero, 0, 0              # $v0 = binom(n-1, k-1) + binom(n-1, k)
    beq $zero, $zero, $zero, $imm1, end_binom, 0 #finish

base_case:
    add $v0, $imm1, $zero, $zero, 1, 0          # Set the result to 1
    beq $zero, $zero, $zero, $imm1, end_binom, 0 #finish


smaller_case:
    add $v0, $zero, $zero, $zero, 0, 0          # Set the result to 0
```

Back to top

```
end_binom:
   lw $a0, $sp, $imm1, $zero, 0, 0          # restore $a0 from stack
   lw $a1, $sp, $imm1, $zero, 1, 0          # restore $a1 from stack
   lw $s0, $sp, $imm1, $zero, 2, 0          # restore $s0 from stack
   lw $ra, $sp, $imm1, $zero, 3, 0          # restore $ra from stack
   add $sp, $sp, $imm1, $zero, 4, 0          # Deallocate space on the stack

   beq $zero, $zero, $zero, $ra, 0, 0        # Return from function in address in $ra
```

circle.asm

תוכנית זו מציירת מעגל על המסך בהתאם לרדיוס נתון, כאשר מרכז המעגל הוא במרכז המסך.
בפונקציה יש חמישה חלקים עיקריים:

1. main – הקצאת במקום במחסנית וטעינת הרדיוס מהזיכרון וכן חישוב הרדיוס בריבוע.
2. לולאת פיקסלים – ישנן שתי לולאות מקוונות למעבר על כל פיקסל במסך. עבור כל פיקסל קוראים לפונקציה שבודקת אם הם בתוך המעגל.
3. בדיקה אם הפיקסל נמצא בתוך המעגל (is_inside) - בחלק זה מחשבים את המרחק מהפיקסל הנוכחי למרכז ע"י פיתגורס. אם המרחק בריבוע קטן או שווה לרדיוס בריבוע, אז ערך ההחזרה הוא 1, אחרת 0. לבסוף חוזרים לפונקציה הקוראת.
4. ציור פיקסל: אם הפיקסל נמצא בתוך המעגל, מוגדרת הכתובת והצבע שלו, והפיקסל מצוייר.
5. סיום התוכנית - לאחר שסיימנו לעבור על כל הפיקסלים התוכנית עוצרת.

```
.word 0x100 50

   sll $sp, $imm1, $imm2, $zero, 1, 11     # set $sp = 1 << 11 = 2048
   add $sp, $sp, $imm1, $zero, -3, 0          # adjust stack for 3 items
   sw $zero, $sp, $imm1, $s2, 2, 0
   sw $zero, $sp, $imm1, $s1, 1, 0
   sw $zero, $sp, $imm1, $s0, 0, 0
   lw $s2, $imm1, $zero, $zero, 0x100, 0       # Load the radius value
   mac $s2, $s2, $s2, $zero, 0, 0            # calculate r^2

 #    center coordinates

   add $s1, $imm1, $zero, $zero, 128, 0                # Initialize x center
   add $s0, $imm1, $zero, $zero, 128, 0                # Initialize y center

   add $t0, $zero, $zero, $zero, 0, 0                # Initialize counter for x


   loop_x:
```

Back to top

```
    add $t1, $zero, $zero, $zero, 0, 0                # Initialize counter for y

loop_y:
    add $t2, $imm1, $zero, $zero, 256, 0              # loop limit
    beq $zero, $t0, $t2, $imm2, 0, end_function               # exit condition
    sub $t2, $s1, $t0, $zero, 0, 0                  # (pixelX - screenCenter)
    sub $a0, $s0, $t1, $zero, 0, 0                  # (pixelY - screenCenter)
    mac $t2, $t2, $t2, $zero, 0, 0                  # Square the result
    mac $a0, $a0, $a0, $zero, 0, 0                  # Square the result
    add $a0, $t2, $a0, $zero, 0, 0                  # Sum of squared values
    ble $zero, $a0, $s2, $imm1, pixel_on, 0           # color if within the radius

adding_y:
    add $t1, $t1, $imm1, $zero, 1, 0                # y += 1
    add $a0, $zero, $imm1, $zero, 256, 0
    bne $zero, $t1, $a0, $imm1, loop_y, 0             # check if should do another y iteration
    add $t0, $t0, $imm1, $zero, 1, 0                # or start a new x
    jal  $ra, $zero, $zero, $imm1, loop_x, 0

pixel_on:
    mac $a1, $t0, $imm1, $zero, 256, 0
    add $a1, $a1, $t1, $zero, 0, 0                  # pixel = 256*x + y
    out $zero, $zero, $imm1, $a1, 20, 0               # set pixel address
    out $zero, $zero, $imm2, $imm1, 255, 21           # set pixel color to white
    out $zero, $zero, $imm2, $imm1, 1, 22       # draw pixel
    jal  $ra, $zero, $zero, $imm1, adding_y, 0


end_function:
    lw $s0, $sp, $imm1, $zero, 0, 0
    lw $s1, $sp, $imm1, $zero, 1, 0
    lw $s2, $sp, $imm1, $zero, 2, 0
    add $sp, $sp, $imm1, $zero, 3, 0                  # pop 3 items from stack
    halt $zero, $zero, $zero, $zero, 0, 0
```

disktest.asm

תוכנית זו מעבירה את תוכן 8 הסקטורים הראשונים בדיסק סקטור אחד קדימה. בתחילת התוכנית שומרים את הסקטור הנוכחי שצריך להעביר ואת סקטור היעד (בתחילה 7,8 בהתאמה). לאחר מכן נקרא מהסקטור המתאים בדיסק לתוך הזיכרון. ואז, נבצע בדיקה של הסטטוס בלולאה עד שהדיסק יהפוך לפנוי כדי שנדע שסיימנו את הקריאה. אחר כך, נעתיק את תוכן הבאפר לסקטור היעד. נחזור על הבדיקה שהדיסק פנוי כדי שנדע שהכתיבה לסקטור היעד התבצעה. לבסוף נחסר מרגיסטרי המקור והיעד אחד ונקפוץ לשלב ביצוע

Back to top

הקריאה מרגיסטר היעד שכן בדקנו קודם לכן שהדיסק פנוי. נמשיך בתהליך עד שנזיז את כל התוכן של 8 סקטורים.

```
main:
   add $t0, $imm1, $zero, $zero, 7, 0                # set source sector to 7 (start from the
last sector)
   add $t1, $imm1, $zero, $zero, 8, 0                # set destination sector to 8
   sll $sp, $imm1, $imm2, $zero, 1, 11               #            set $sp = 1 << 11 = 2048

move_sectors:
 #   Read sector content from the current sector
   out $zero, $zero, $imm1, $t0, 15, 0               # set sector
   out $zero, $zero, $imm1, $imm2, 16, 0               # set buffer
   out $zero, $zero, $imm1, $imm2, 14, 1               # set diskcmd to read

check_status:
   in $t2, $zero, $imm1, $zero, 17, 0                # check the disk status
   bne $zero, $t2, $zero, $imm1, check_status, 0         # check until status is 0

 #   Write sector content to the destination sector
   out $zero, $zero, $imm1, $t1, 15, 0               # set sector
   out $zero, $zero, $imm1, $imm2, 14, 2               # set diskcmd to write

check_status2:
   in $t2, $zero, $imm1, $zero, 17, 0                # check the disk status
   bne $zero, $t2, $zero, $imm1, check_status2, 0         # check until status is 0

 #   Move to the previous sectors
   add $t0, $imm1, $t0, $zero, -1, 0                 # decrement source sector
   add $t1, $imm1, $t1, $zero, -1, 0                 # decrement destination sector


 #   Check if we have moved all 8 sectors
   bge $zero, $t0, $zero, $imm1, move_sectors, 0         # branch if source sector is greater
than or equal to zero

   Halt $zero, $zero, $zero, $zero, 0, 0               # Halt the system
```

Back to top