פרויקט 1 במבנה המחשב

מגישים:

תום שחר – 311494066

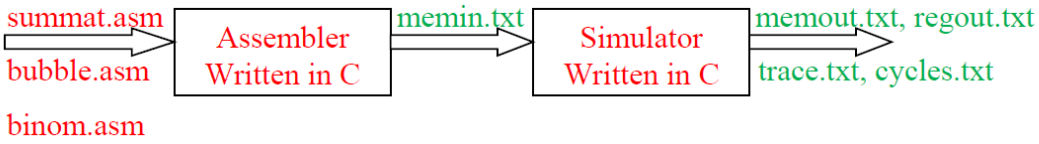
ניר קקון – 312203672

אדיר לייטמן – 206828881

תיאור כללי

מטרת הפרויקט הייתה לתרגל את נושאי שפת המחשב, קלט / פלט, וכמו כן תרגול יכולות התכנות בשפת C. בפרויקט מימשנו אסמבלר וסימולטור (שתי תוכניות נפרדות). בנוסף, כתבנו תכניות בשפת אסמלי עבור מעבד RISC בשם SIMP, אשר דומה למעבד MIPS אך פשוט ממנו.

הדיאגרמה הבאה ממחישה את רכיבי הפרויקט:



- שלושת הקלטים הרשומים באדום מציינים את תכניות האסמבלי שנדרשנו לכתוב.
- מימוש תוכנית האסמבלר בשפת C.
- קובץ memin (כתוב בירוק) הינו הפלט של קובץ האסמבלר ומהווה קלט בעבור הסימולטור שלנו.
- מימוש הסימולטור בשפת C.
- ארבעה קבצי פלט שהסימולטור מוציא עליהם נרחיב בהמשך.

האסמבלר אותו מימשנו קורא את תוכנית האסמבלי שנטענה לו כקלט (אחד משלושת הקלטים) ומבצע תרגום שלה לשפת מכונה. הקובץ המתורגם נפלט החוצה ומהווה עבור הסימולטור את מסד הנתונים עבורו – הזיכרון.

הסימולטור קורא את קובץ הזיכרון שכתוב בשפת מכונה הכולל את הפקודות וה-Data שנשמר בו ומבצע על בסיסו את הפקודות. בסיום הפעולה הוא פולט את הקבצים אשר מתעדים ומפרטים את פעולותיו.

הסימולטור יסמלץ את מעבד ה - SIMP , כאשר כל הוראה במעבד מתבצעת במחזור שעון אחד. המעבד עובד בתדר של 1024 הרץ - מבצע 1024 הוראות אסמבלי בשנייה.

רגיסטרים

מעבד SIMP מכיל 16 רגיסטרים, שכל אחד מהם ברוחב 32 ביטים. שמות הרגיסטרים, מספרם, ותפקיד כל אחד מהם בהתאם ל - calling conventions , נתונים בטבלה הבאה:

Register Number	Register Name	Purpose
0	$zero	Constant zero
1	$imm	Sign extended immediate
2	$v0	Result value
3	$a0	Argument register
4	$a1	Argument register
5	$t0	Temporary register
6	$t1	Temporary register
7	$t2	Temporary register
8	$t3	Temporary register
9	$s0	Saved register
10	$s1	Saved register
11	$s2	Saved register
12	$gp	Global pointer (static data)
13	$sp	Stack pointer
14	$fp	Frame Pointer
15	$ra	Return address

סט הוראות וקידודן

למעבד SIMP יש פורמט בודד לקידוד כל ההוראות. כל הוראה הינה ברוחב 32 ביטים, כאשר מספרי הביטים של כל שדה נתונים בטבלה הבאה:

31:24	23:20	19:16	15:12	11:0
Opcode	rd	rs	rt	immediate

הפקודות הנתמכות ע"י המעבד ומשמעות כל פקודה נתונים בטבלה הבאה:

Opcode Number	Name	Meaning
0	add	R[rd] = R[rs] + R[rt]
1	sub	R[rd] = R[rs] – R[rt]
2	and	R[rd] = R[rs] & R[rt]
3	or	R[rd] = R[rs] | R[rt]
4	sll	R[rd] = R[rs] << R[rt]
5	sra	R[rd] = R[rs] >>> R[rt], arithmetic shift with sign extension
6	srl	R[rd] = R[rs] >> R[rt], logical shift
7	beq	if (R[rs] == R[rt]) pc = R[rd]
8	bne	if (R[rs] != R[rt]) pc = R[rd]
9	blt	if (R[rs] < R[rt]) pc = R[rd]
10	bgt	if (R[rs] > R[rt]) pc = R[rd]
11	ble	if (R[rs] <= R[rt]) pc = R[rd]
12	bge	if (R[rs] >= R[rt]) pc = R[rd]
13	jal	R[15] = pc + 1 (next instruction address), pc = R[rd]
14	lw	R[rd] = MEM[R[rs]+R[rt]]
15	sw	MEM[R[rs]+R[rt]] = R[rd]
16		Reserved for future use
17		Reserved for future use
18		Reserved for future use
19	halt	Halt execution, exit simulator

בנוסף, בפרויקט זה מוגדרת פקודת Pseudo code הנקראת "word." בפורמט הבא:

.word address data

כאשר address הינו כתובת המילה ו-data הינו תוכן המילה, כל אחד מהשדות יכול להיות בפורמט דצימלי או הקסאדצימלי.

מידע והנחות נוספות

1. הזיכרון מכיל 512 מילים כאשר כל מילה ברוחב של 32 סיביות.
2. משתנה ה-(Program Counter (PC כמו גם כתובות הזיכרון יתקדמו ב-1 ולא ב-4.
3. ניתן להניח שאורך השורה המקסימאלי בקבצי הקלט הוא 500.
4. ניתן להניח שאורך ה - label המקסימאלי הוא 50.
5. פורמט ה - label מתחיל באות, ואח"כ כל האותיות והמספרים מותרים.
6. צריך להתעלם מ - whitespaces כגון רווח או Tab. מותר שיהיו מספר רווחים או Tabs ועדיין הקלט נחשב תקין.
7. יש לתמוך בספרות הקסאדצימליות גם ב - lower case וגם upper case.

אסמבלר

כללי:

תוכנית האסמבלר היא תוכנית שמטרתה לקרוא קבצי אסמבלי, לפענח את תוכנם ולתרגם אותם לשפת מכונה (ספרות הקסאדצימליות). בתוכנית נעזרנו בקבצי header (ספריות) שסייעו לנו בשמירה על הסדר והנראות של הקוד.

קלט:

האסמבלר מקבל כקלט תוכנית אסמבלי הכתובה בקובץ asm.

פלט:

קובץ טקסט בשם memin המהווה את תמונת הזיכרון בעבור הסימולטור – כלומר את הפקודות לביצוע וה-Data השמור בזיכרון.

קבצי הרצה:

Assembler.c – קובץ ראשי המכיל את פונקציית ה-main שקוראת לפונקציות עזר המבצעות את פעולתו של רכיב האסמבלר, פונקציות אלו ממומשות באמור בספריות.

- **פונקציה ראשית:**

int main(int argc, char* argv[]) - פונקציית הפותחת את קובץ הקלט (אסמבלי) וקוראת לפונקציות אשר מממשות את אופן פעולת הרכיב ע"י שני מעברים על הקלט, שתי "ריצות", ולבסוף מדפיסה את תוכן הזיכרון בקובץ text בשם memin אשר מהווה קלט לסימולטור.

- **ספריות:**

1. **First _run** - מכילה את כלל פונקציות העזר הנדרשות עבור הריצה הראשונה, מטרתן לאתר את ה-Labels ולשדך את ה-PC המתאים לכל אחד מהם. Label (תווית) הוגדרה כמבנה הכולל שם ו-PC, כלל התוויות שנקראו מהסימולציה נשמרות במבנה נתונים של רשימה מקושרת. להלן המבנה מבנה העזר אשר ישמש כרשימה של התוויות:

```
// Label struct

// 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.
typedef struct Label {
    char name[MAX_LABEL_LEN];// name will store the name of the label
    int address; // address will store the line number (pc) and will be the immediate value in related jump and branch commands
    struct Label* next;
} label;
```

<u>פונקציות העזר הממומשות בספריה זו הן:</u>

1) **label* create_label(char* name, int pc)**
קלט: שם ומיקום במערך הזיכרון (PC).
פעולה: מקצה זיכרון ובונה את מבנה התווית.
פלט: מצביע לתווית שנוצרה.

2) **label* add_label_to_list(label* head, char* name, int PC)**
קלט: שם של תווית, מצביע לראש רשימת התוויות ואת ערך ה-PC של התווית.
פעולה: קוראת ל **create_label** אשר מייצרת תווית חדשה ומוסיפה את התווית לרשימה המקושרת.
פלט: מצביע לאיבר החדש שנוסף (ראש הרשימה כעת).

3) **void free_label_list(label* head)**
קלט: מצביע לראש הרשימה.
פעולה: משחררת את הרשימה.
פלט: ללא.

4) **int also_instruction(char* line)**
קלט: שורה מקובץ האסמבלי.
פעולה: ייצור אינדיקטור לשורה המסמן אם היא מכילה פקודה לביצוע.
פלט: 0 עבור שורה ללא פקודה לביצוע, 1 עבור שורה עם פקודה לביצוע.

5) **label* first_run(FILE *assembly)**
קלט: מצביע לקובץ אסמבלי
פעולה: קריאת הטקסט (תוכן קוד האסמבלי), "ניקוי" הטקסט מתווים שאינם רלוונטיים (באמצעות שימוש בפונקציות עזר נוספות) והבחנה בין תוויות, פקודות וכו'.
פלט: רשימה מקושרת של התוויות ע"פ המבנה שהוזכר לעיל.

6) **void LableChange(Memory_Line_list* memory_list, label* label_list)**
קלט: מצביע לרשימה המקושרת של התוויות ומצביע לרשימה המקושרת של הפקודות.
פעולה: שינוי שם התווית לערך ה-PC שלה ובמקרה של '$zero' שינוי ערכו ל0.
פלט: ללא.

7) **int find_label(label* label_list, char* label_to_find)**
קלט: שם של תווית רצויה ומצביע לראש הרשימה של התוויות.
פעולה: עוברת על רשימת התוויות ומחפשת את התווית הרצויה.
פלט: כתובת התווית המבוקשת (במובני PC).

8) **void clean_line(char* line)**
קלט: שורה מקובץ האסמבלי.
פעולה: קוראת לשתי פונקציות המסירות תווים לבנים והערות.
פלט: ללא.

9) **void clean_comment(char* line)**
קלט: שורה מקובץ האסמבלי.
פעולה: הסרת הערות (#).
פלט: ללא.

10) **void clean_white_spaces(char* line)**
קלט: שורה מקובץ האסמבלי.
פעולה: הסרת תווים לבנים.
פלט: ללא.

2. **second _run** - ספריה המכילה את כלל פונקציות העזר הנדרשות עבור הריצה השנייה, מטרתה לבצע תרגום של שורות הפקודות מקובץ האסמבלר לשפת מכונה (הקסדצימל) ושמירתן ברשימה מקושרת, רשימה זו מכילה את הזיכרון שיודפס בהמשך לקובץ ה-memin ויהווה הזיכרון ההתחלתי לפני ביצוע הפקודות המפורטות בו. כל שורת פקודה מועברת כמבנה בודד בתוך רשימת הזיכרון המכיל את כל שדות הפקודה (rd,op,imm,rt,rs) ויוצרת רשימה מקושרת לכל ההוראות.
להלן שני מבני הנתונים בהם נעשה שימוש – Memory_Line_list ישמש כרשימה שמחזיקה את הוראות הקוד ו-Memory המחזיק את כתובת תחילת רשימת הזיכרון וכתובת סוף הרשימה – הPC בעל הערך הגבוהה ביותר.

```
// this struct will be used to save the memory lines
typedef struct Memory_Line_list {
    char opcode[6];// opcode name can be 5 characters and '/0' to  8 bit
    char rd[6];    //  reg name can be 5 characters and '/0' to a 4 bit
    char rs[6];    //  reg name can be 5 characters and '/0' to a 4 bit
    char rt[6];    //  reg name can be 5 characters and '/0' to a 4 bit
    char imm[51];  //  immediate value or label name can be 50 characters and '/0' to 12 bit
    int pos_pc;    // position of character in line
    struct Memory_Line_list* next;
}Memory_Line_list;


// Memory struct and related functions.
//it is used so the --second iteration-- can return two values.
typedef struct Memory {
    Memory_Line_list* head;// head of memory line list
    int last;// the position of the last line (pc) in the memory
}Memory;
```

<u>פונקציות העזר הממומשות בספריה זו הן:</u>

1) **Memory_Line_list* create_memory_line(char* opcode, char* rd, char* rs, char* rt, char* imm, int current_pos)**
קלט: כל השדות הרלוונטיים להוראה ובנוסף את ערך ה-PC שלה.
פעולה: יוצרת ומאתחלת איבר שבהמשך יוכנס לרשימה המקושרת.
פלט: מצביע לאיבר זה.

2) **Memory_Line_list* add_line_to_memory_list(Memory_Line_list* head, char* opcode, char* rd, char* rs, char* rt, char* imm, int address_pos)**
קלט: של הוראה, ערך PC ומצביע לראש הרשימה.
פעולה: מוסיפה את האיבר לסוף הרשימה.
פלט: מצביע לראש הרשימה.

3) **Memory* create_mem(Memory_Line_list* head, int pos1)**
קלט: מצביע לראש רשימת הזיכרון וכתובת PC של האיבר האחרון.
פעולה: יוצרת מבנה של זיכרון שמחזיק את כתובת תחילתה של הרשימה המקושרת וערך PC האחרון בכל הקוד.
פלט: מצביע אל המבנה.

4) **void free_memory_Line_list(Memory_Line_list* head)**
קלט: מצביע לראש הרשימה.
פעולה: משחררת את הזיכרון עם תום השימוש.
פלט: ללא.

5) **void free_memory_struct(Memory* mem)**
קלט: מצביע למבנה הזיכרון.
פעולה: פונקציה המשחררת את הזיכרון בעבור מבנה הנתונים ורשימת הזיכרון.
פלט: ללא.

6) **int string_to_int(char* string)**
קלט: מחרוזת של תווים.
פעולה: ממירה את המחרוזת למספר, הפונקציה ממירה שני סוגי מחרוזות - אחת המייצגת ספירה עשרונית והשנייה ספירה הקסהדצימלית.
פלט: מספר לאחר ההמרה.

7) **Memory_Line_list *WordCommand(Memory_Line_list* head, char line[MAX_LINE], int *last_pc)**
קלט: מצביע לראש הרשימה, שורה וערך ה-PC האחרון הנוכחי.
פעולה: יוצרת איבר ברשימה המקושרת של הוראות word. ויוצרת שם איבר באופן דומה ליתר ההוראות. הפונקציה מזינה רק את שדה כתובת היעד והמידע בהתאם לפקודה word. ועבור שאר השדות מזינה את הערך "blank" המשמש כ FLAG.
פלט: מצביע לראש הרשימה של הזיכרון, הרשימה מעודכנת עם השורה החדשה.

8) **void read_opcode(char *line[MAX_LINE], char *opcode)**
קלט: שורת מידע ומחרוזת של opcode.
פעולה: מעתיקה את ה-opcode לתוך השורה.
פלט: ללא.

9) **void readDollar(char* line[MAX_LINE])**
קלט: שורה מקובץ האסמבלר.
פעולה: "רצה" על השורה עד הופעת סימן $ ומעדכנת את המצביע של השורה למיקום תו הדולר.
פלט: ללא.

10) **void read_register_name(char* line[MAX_LINE], char* register_name)**
קלט: שורה ומחרוזת ריקה.
פעולה: מעתיקה את שם הרגיסטר מהשורה למחרוזת הריקה.
פלט: ללא.

11) **void readImmd(char *line[MAX_LINE], char *imm)**
קלט: שורה ומחרוזת ריקה.
פעולה: קוראת את ערך ה-imm ומעתיקה אותו למחרוזת ה-imm.
פלט: ללא.

12) **Memory_Line_list *readLine(char *line, int *last_line, int *current_position, Memory_Line_list *head)**
קלט: שורה, מיקום השורה האחרונה (במונחי PC), PC נוכחי ומצביע לראש הרשימה.
פעולה: קוראת את השורה, בונה איבר חדש ע"פ השורה ומוסיפה איבר זה לרשימה המקושרת.
פלט: מצביע לראש הרשימה המקושרת.

13) **Memory* SecondRun(FILE* assembly)**
קלט: קובץ האסמבלי.
פעולה: קריאת הטקסט, העברת שורות הפקודה לרשימה מקושרת של פקודות ובניית מבנה של זיכרון.
פלט: מבנה נתונים של הזיכרון.

3. **Output –** ספריה המכילה בתוכה פונקציות עזר שמטרתן להדפיס את תוכן ה-memin לקובץ טקסט, זהו המצב ההתחלתי של הזיכרון לפני ביצוע הפקודות בסימולטור. הדפסה. הקובץ מכיל בתוכו פקודות עם התניות מרובות לפי תווים ייעודיים שמאפיינים סוגי פקודות, רגיסטרים וסימונים נוספים.
בקובץ זה יצרנו שני מבני ENUM אשר מסייעים בהוראות הפלט:

```
enum OPCODE {
    add, sub, and, or , sll, sra, srl, beq, bne, blt, bgt, ble, bge, jal, lw, sw, Res1, Res2, Res3, halt,
};

enum REGISTER {
    $zero, $imm, $v0, $a0, $a1, $t0, $t1, $t2, $t3, $s0, $s1, $s2, $gp, $sp, $fp, $ra,
};
```

<u>פונקציות העזר הממומשות בספריה זו הן:</u>

1) **void printRegisterNumber(char *register_name, FILE *memin, int *data)**
קלט: שם הרגיסטר, קובץ ה-memin ומספר המציין את המידע.
פעולה: מכילה מספר רב של התניות הבודקות האם שם הרגיסטר זהה לאחד מהשמות המופיעים בטבלאות המפורטות תחת "רגיסטרים". בהתאם לתשובה הפונקציה מדפיסה את הערכים בקובץ ה-memin.
פלט: ללא.

2) **int printOpcode(char *opcode, FILE *memin)**
קלט: opcode וקובץ ה-memin.
פעולה: הדפסת ה-opcode הרלוונטי, אם קיים, לקובץ.
פלט: אינדיקטור 1 במידה והשורה כללה opcode, 0 אחרת (word.).

3) **void print_data_file(Memory* mem, FILE *memin)**
קלט: רשימה מקושרת של הזיכרון ומצביע לקובץ memin.
פעולה: מדפיסה את הרשימה המקושרת של הזיכרון לקובץ ה-memin בפורמט הנדרש.
פלט: ללא.

סימולטור

כללי:

תפקיד הסימולטור הוא לקרוא את קובץ הזיכרון memin אותו הוא מקבל כקלט, לבצע את הפקודות המפורטות בו ולעדכן את תמונת הזיכרון בהתאם. תמונת הזיכרון הסופית, memout, מיוצרת כקובץ פלט לצד 3 קבצים נוספים המפורטים מטה.

קלט:

קובץ memin.txt הנפלט מהאסמבלר ומהווה את הזיכרון של המעבד לפני ביצוע הפקודות.

פלט:

ארבעה קבצי טקסט המתעדים את פעולת המעבד:

- Memout – תמונת הזיכרון לאחר מימוש כל הפעולות.
- Regout – קובץ המכיל את תוכן הרגיסטרים R2-R15 בסיום הריצה.
- Trace – קובץ המתעד כל פעולה שבוצעה בסימולטור ע"י עדכון ערכי הרגיסטרים לפני ביצוע ההוראה. הערכים מופיעים כשורות מידע בפורמט הקסאדצימלי בסדר הבא:
  PC INST R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 R15
- Cycles – קובץ טקסט המציג את מספר מחזורי השעון שנדרש לסימולטור לבצע את הפעולות כמפורט ב-memin (ובמקור בקובץ האסמבלי).

קבצי הרצה:

Simulator.c – קובץ ראשי המכיל את פונקציית ה-main ופונקציות עזר המבצעות את פעולתו של רכיב האסמבלר, פונקציות אלו ממומשות בקובץ הנ"ל ללא ספריות נוספות.

פונקציה ראשית:

int main(int argc, char* argv[]) – הפונקציה קוראת לפונקציות העזר באופן הבא:

- קריאה לפונקציה ה קוראת את קובץ ה-memin ומייצרת את ארבעת קבצי ה-txt של הפלטים המפורטים מעלה.
- קריאה לפונקציית fetch על מנת לקרוא את תמונת הזיכרון
- מבצעת לולאה בהתאם לPC הנוכחי (מתחילה ב PC=0) – בכל איטרציה מבתצעת בנייה של מבנה הפקודה ע"י מבנה עזר, מודפסים הערכים הרלוונטיים לקובץ ה- trace ולבסוף מתבצעת הפעולה ב-PC הנוכחי תוך קידום מחזור השעון.
- בסיום הריצה שלושת הקבצים הנותרים (Memout, Regout, Cycles) מודפסים.
- מתבצעת קריאה לפונקציה הסוגרת את הקבצים.

להלן מבנה העזר לייצוג פקודה:

```
typedef struct Format
{
    int opcode; // 8 bits
    int rd; // 4 bits
    int rs; // 4 bits
    int rt; // 4 bits
    int imm; // 12 bits.
}Format;
```

פונקציות העזר:

1. **int fetch(FILE* txt_file)**

קלט: מצביע לקובץ memin.txt.

פעולה: קריאת תמונת הזיכרון שנכנסת כקלט לסימולטור והעברת המידע לרשימה מקושרת אשר עליה הסימולטור ייבצע את הפעולות, רשימה זו תכיל את הנתונים אשר יודפסו Memout בסיום הפונקציה הקראשית.

פלט: מספר המייצג את מספר המילים הנמצאים בתמונת הזיכרון הנכנסת (ה-PC האחרון של הזיכרון).

2. **void build_instruction(int PC, Format* ptr_inst)**

קלט: PC ומצביע למבנה המייצג פקודה.

פעולה: קריאת הזיכרון בהתאם לPC, המרת הערכים להקסאדצימלי ובניית איבר במבנה העזר ע"פ מבנה פקודת I-Format:

31:24	23:20	19:16	15:12	11:0
Opcode	rd	rs	rt	immediate

פלט: ללא.

3. **int execute(Format* ptr_inst)**

קלט: פוינטר למיקום הפקודה הנוכחית.

פעולה: קריאת ערכי הפקודות שמופיעות בכל מילה בזיכרון וביצוע הפעולה בהתאם ל-opcode כמפורט בטבלה תחת "סט הוראות וקידודים".

פלט: אינדיקטור 1 אם בוצעה קפיצה למקום אחר בזיכרון אשר דרש עדכון פרטני של PC, 0 אחרת (PC יגדל ב-1 בפונקציה הראשית).

4. **void open_files(char* argv[], FILE** memin, FILE** memout, FILE** trace, FILE** cycles, FILE** regout)**

קלט: מצביע לכתובות של הקבצים.

פעולה: פותחת את הקבצים הקלט והפלט איתם נעבוד.

פלט: ללא.

5. **void close_files(FILE** memin, FILE** memout, FILE** trace, FILE** cycles, FILE** regout)**

קלט: מצביעים לכתובות הקבצים.

פעולה: סגירת קבצי הטקסט איתם עבדנו.

פלט: ללא.

תוכניות בדיקה (אסמבלי)

Summat:

תוכנית הממשת חיבור של שתי מטריצות. כתובות הזיכרון הספציפיות מוקצות בהתאם לדרישה (0x100-0x11F) ע"י פקודת "word.". התוכנית רצה בלולאה עם מונה בהתאם למספר הערכים אותם מחברים (סה"כ 16 איברים בכל מטריצה). מבנה המטריצות הוא:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Bubble:

תוכנית הממשת מיון של מערך מספרים בסדר עולה ע"י שימוש באלגוריתם bubble sort. מערך המספרים שאותו יש למיין נמצא בתאים 0x100 עד 0x10F .
התוכנית מכילה שתי לולאות (פנימית וחיצונית) אשר רצות ע"ג המערך ומבצעות פעולת SWAP במידה והערך השמאלי גדול יותר מזה שמימינו, ולא מבצעת במידה ולא. הלולאה מסתיימת כאשר המערך ממוין מקטן לגדול לאחר שהערכים הגבוהים "ביעבעו" למקומם.

16	15	14	11	12	13	9	7	9	6	9	4	3	-1	5	-2

↓

-2	-1	3	4	5	6	7	9	9	9	11	12	13	14	15	16

Binom:

תוכנית הממשת באופן רקורסיבי את מקדם הבינום של ניוטון. גם כאן הכתובות ניתנו מראש והנחת העבודה הייתה כי n מספיק קטן על מנת שלא ייווצר מצב של overflow. התוכנית מומשת ע"פ הנוסחה שניתנה בהנחיות הפרויקט:

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

אופן הרצת הרכיבים

להלן הפעולות שנדרש לבצע על מנת שהקלטים יקלטו בקבצי ההפעלה:

- יש להעביר את קובץ תוכנית הבדיקה (אסמבלי) לתיקיית ה-Assembler. יש להגדיר את שם הקובץ בתוך הפרויקט תחת Properties -> Debugging -> Command arguments ובנוסף את שם קובץ הפלט memin.txt.
- לאחר הרצת האסמבלר יש להעביר את קובץ memin לתיקיית ה-Simulator. קבצי הפלט יופיעו בתיקייה לאחר הרצתו.