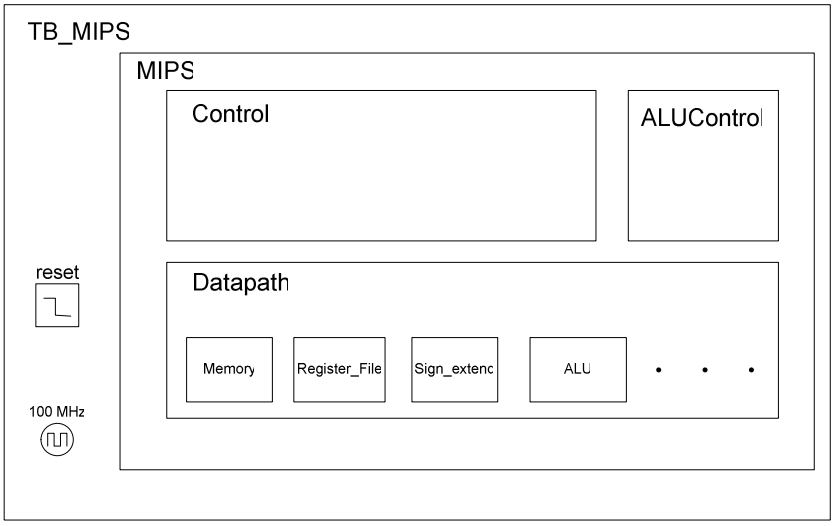
044262 תכן לוגי

סימולציה שלישית ואחרונה!

המעבד שלנו:



הזיכרון שלנו (התוכנית שלנו):

```
[0x00000000]   0x341d0064            ori     $sp,    $zero,  0x64
[0x00000004]   0x8fa40000            lw      $a0,    0($sp)
[0x00000008]   0x1c800004            bgtz    $a0,    init
[0x0000000c]   0x0490000e            bltzal  $a0,    sign
[0x00000010]   0x10800001            beq     $a0,    $zero,  Finish
[0x00000014]   0x1c800001            bgtz    $a0,    init
[0x00000018]   0x08000006  Finish:   j       Finish
[0x0000001c]   0x20050002  init:     addi    $a1,    $zero, 0x2
[0x00000020]   0x20080000            addi    $t0,    $zero, 0x0
[0x00000024]   0x20090001            addi    $t1,    $zero, 0x1
[0x00000028]   0x01285020  fib :     add     $t2,    $t1,    $t0
[0x0000002c]   0x00094020            add     $t0,    $zero,  $t1
[0x00000030]   0x000a4820            add     $t1,    $zero,  $t2
[0x00000034]   0x20a50001            addi    $a1,    $a1,    0x1
[0x00000038]   0x00a4802a            slt     $s0,    $a1,    $a0
[0x0000003c]   0x1600fffa            bne     $s0,    $zero,  fib
[0x00000040]   0xafaa0004            sw      $t2,    4($sp)
[0x00000044]   0x08000006            j       Finish
[0x00000048]   0x3806ffff  sign:     xori    $a2,    $zero,  0xFFFF
[0x0000004c]   0x00063400            sll     $a2,    $a2,    16
[0x00000050]   0x34c6ffff            ori     $a2,    $a2,    0xFFFF
[0x00000054]   0x00862026            xor     $a0,    $a0,    $a2
[0x00000058]   0x20840001            addi    $a0,    $a0,    0x1
[0x0000005c]   0x03e00008            jr      $ra
[0x00000060]   0x00000000
[0x00000064]   0x00000008
[0x00000068]   0x00000000
[0x0000006c]   0x00000000
```

סעיף א

שאלה: עבור אילו פקודות הייתם צריכים להוסיף תמיכה במעבד?

תשובה: יש להוסיף תמיכה לכל הפקודות שהן לא מהרשימה הבאה: lw, sw, add, sub, or, and, slt, beq, j.

סעיף ב

מטלה: הסבירו במפורש את המימוש שלכם (שרטוטים,דיאגראמת מצבים, אותות בקרה ורכיבים שהוספתם).

פתרון: להלן הטבלאות המסכמות את הנושא. בנוסף, מצורפים בסוף שרטוטי דיאגרמת המצבים והמעבד כולו.

טבלה 1: סיכום כל השינויים שיש לבצע כדי לממש כל פקודה.

פקודה חדשה	פורמט הפקודה	שינוי במסלול הנתונים	שינוי בבקר/קוי בקרה
ori	ori rt, rs, imm 0xd | rs | rt | imm	הוספת יחידת Zero-Extention	הוספת קו בקרה Extend לשליטה בבורר הבוחר בין יציאת ה Zero Extention ליציאת ה Sign Extention; יש להרחיב את הקו ALUOp כדי שנוכל לתת פקודת or ל ALU
bgtz	bgtz rs, label 0x7 | rs | 0 | offset	יש לשכלל את ה ALU כך שיוציא חיווי Positive שיהיה גבוה אם התוצאה חיובית ממש; הוספת בורר קפיצה שנשלט ע"י PCWriteCond	שכלול PCWriteCond כך שיבחר איך לקפוץ מהבורר.
bltzal	bltzal rs, label 0x7 | rs | 0x10 | offset	יש לשכלל את הבורר בכניסה ל register file שבוחר מה לכתוב, כדי שיהיה אפשרי לכתוב את תוכן ה PC לאוגר $ra; יש להוסיף בורר בכניסה ל Read Register 2 כדי לאפשר קריאה של אוגר $0;	יש להרחיב את MemtoReg ל 2 ביטים; יש להוסיף את ReadSrc כדי לבחור קריאה של $0; נבחר PCWriteCond=1
addi	addi rt, rs, imm 0x8 | rs | rt | imm	אין שינוי	
bne	bne rs, rt, label 0x5 | rs | rt | offset	אין שינוי	PCWriteCond=2
xori	xori rt, rs, imm 0xe | rs | rt | imm	להוסיף תמיכה ב ALU	יש להרחיב את הקו ALUOp כדי שנוכל לתת פקודת xor ל ALU
xor	xor rd, rs, rt 0 | rs | rt | rd | 0 | 0x26	להוסיף תמיכה ב ALU	אין שינוי
jr	jr rs 0 | rs | 0 | 8	להוסיף חיבור מאוגר A לבורר שבוחר מה לכתוב ל PC, וכן להרחיב בורר זה.	יש להכניס לבקר גם את סיביות Instruction[0-5], כדי שנוכל להתפצל על-סמך ה funct.
sll	sll rd, rt, shamt 0 | rs | rt | rd | shamt | 0	להוסיף תמיכה ב ALU; להכניס את Instruction[6-10] לתוך ה ALU.	אין שינוי

טבלה 2: פירוט כל השינויים שיש לבצע במעבד

מה משנים	איפה משנים	איך משנים
יחידת Zero_extend_ins	מסלול נתונים	כניסות: INS[0-15] יציאות: extender_mux_signal
בורר Extender_MUX בין Sign Extend ל Zero Extend	מסלול נתונים	כניסות: zero_extended, sign_extended בקרה: Extend יציאה: extender_mux_signal
קו בקרה Extend עבור הבורר הנ"ל	בקר ומסלול נתונים	תוספת
אות extender_mux_signal מהבורר החדש Extender_MUX לכניסה 2 בבורר ALUSrcB_MUX	מסלול נתונים	תוספת
יציאה חדשה Positive	ALU	
בורר Branch_MUX , בורר 8 ←1. בורר זה יטפל בסוגי הקפיצות המותנות השונות.	מסלול נתונים	כניסות: zero , not(zero), positive, nor(positive,zero), אפס קבוע. יציאה: BranchType בקרה: PCWriteCond
הערך ש PCLoad מקבל	בקר	הערך החדש: PCWRite OR BranchType
PCWriteCond. על קו זה הבקר יוציא את סוג הקפיצה המותנית.	בקר	הרחבה ל 3 סיביות
ביטול Zero ככניסה לבקר	בקר	Zero יהיה כעת אות שבנוסף לאות Positive יעזור לחשב האם לקפוץ
BranchType יכנס לבקר במקום האות Zero שביטלנו. אות זה יכיל את האינפורמציה האם לקפוץ	בקר ומסלול נתונים	
שכלול הבורר WriteData_MUX ל 4 ← 1	מסלול נתונים	כניסה 3: PC בקרה: MemtoReg
שכלול הבורר WriteReg_MUX ל 4 ← 1	מסלול נתונים	כניסה 3: קבוע 31, כדי לכתוב את תוכן ה PC לאוגר $ra בפקודת bltzal בקרה: Regdst
קו בקרה Regdst	בקר ומסלול נתונים	הרחבת הקו ל 2 סיביות
קו בקרה MemtoReg	בקר	הרחבת הקו ל 2 סיביות
הוספת בורר ReadReg2_MUX, גודלו 2 ← 1	מסלול נתונים	כניסה: Inst[16-20], אפס קבוע יציאה: ReadReg2 בקרה: ReadSrc
קו בקרה ReadSrc לבורר הנ"ל	בקר	
כניסה חדשה לבקר - funct	בקר	
חיבור יציאת רגיסטר A לכניסת הבורר PC_MUX	מסלול נתונים	

טבלה 3: קודי ALUOp ו ALUCtrl המעודכנים לתמיכה בכל הפקודות הנדרשות

ALUOp	ALUCtrl	פקודה
000	010	חיבור
001	110	חיסור
100	100	xor
101	001	or
110	000	and
	011	sll
010	לפי שדה funct	

סעיף ג

מטלה: הריצו סימולציה עם התוכנית הנ"ל. עליכם להגיש דיאגראמות גלים (waveforms) אשר ידגימו ביצוע נכון של התוכנית (הוסיפו לצורת הגלים אותות אותם אתם מוצאים לנכון להוסיף על מנת להשיג מטרה זו; לדוגמה, עקבו אחר עדכון הרגיסטר $t0).

פתרון:

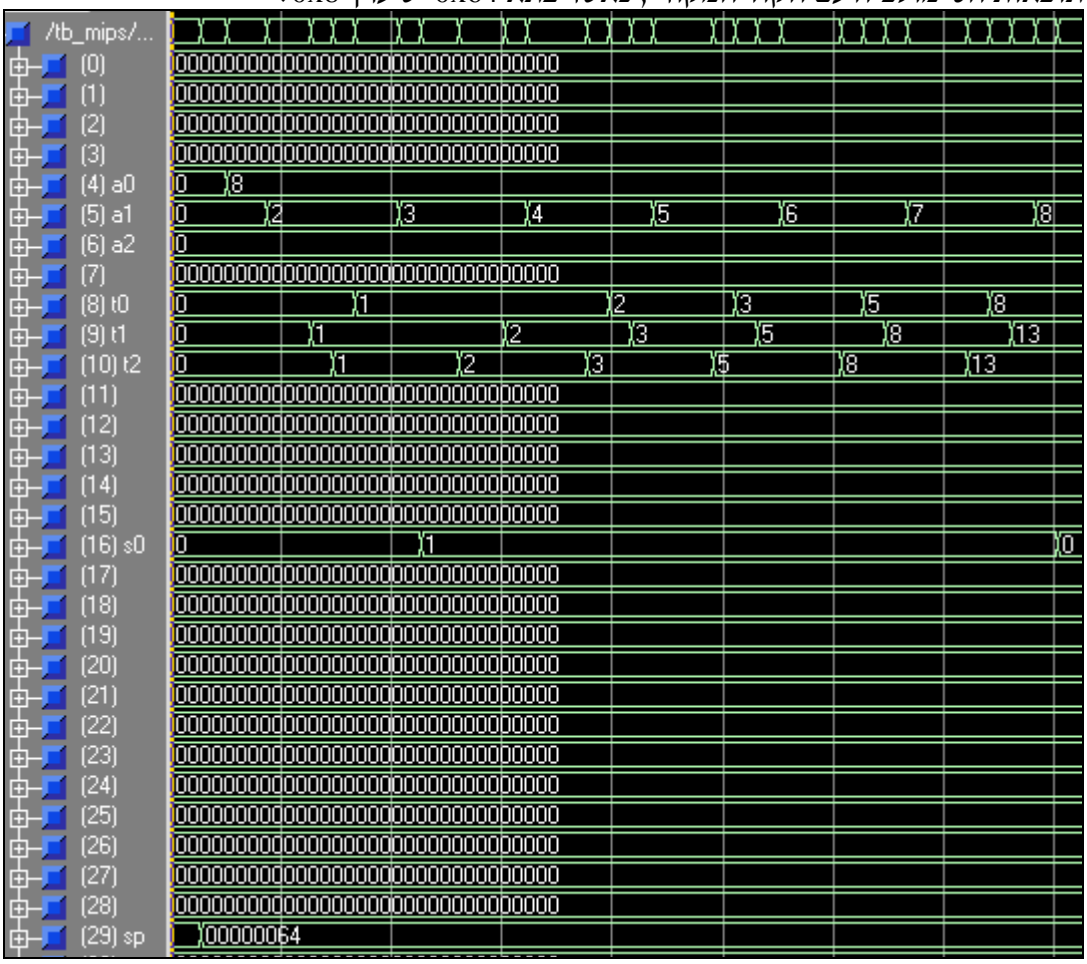
להבנת התוכנית, נתרגמה לשפה עילית:

```
sp = 0x64
a0 = mem[sp]
if (a0 < 0) {
     a1 = -a1
}
if (a0 > 0) {
     a1 = 2
     t0 = 0
     t1 = 1
     do {
          t2 = t0 + t1
          t0 = t1
          t1 = t2
          a1 = a1 + 1
     } while (a1 < a0)
     mem[sp+4] = t2
}
```

ובכמה מילים:

התוכנית קוראת את הערך שרשום בזיכרון בתא 0x64. הערך המוחלט של ערך זה יהיה מספר האיבר בסדרת פיבונאצ'י (0,1,1,2,3,5,8,13,21,...) שהתוכנית תחזיר בתא הזכרון מספר 0x68.

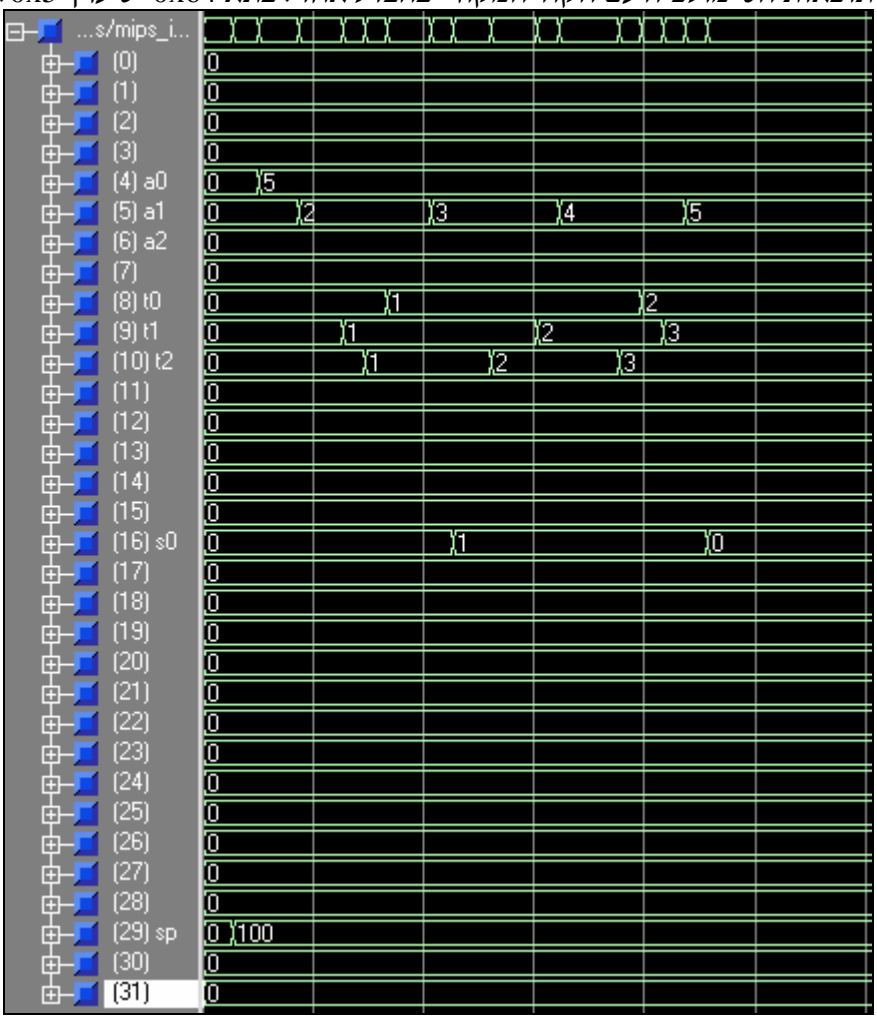
תוצאות הסימולציה עם הקוד המקורי, כאשר בתא 0x64 יש ערך 0x8:



מתבצעת יציאה מהלולאה הראשית כאשר s0 מקבל את הערך 0.

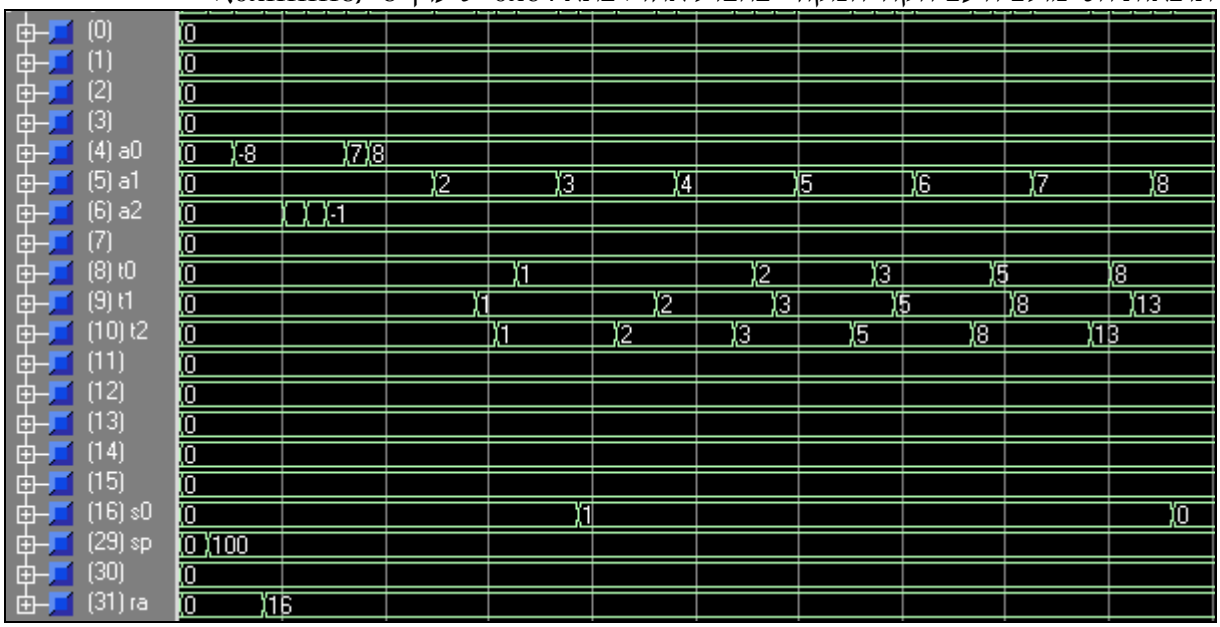
הערך שיכתב לתא 0x68 בזיכרון יהיה הערך שבאוגר t2, וזהו 13, האיבר השמיני בסדרת פיבונאצ'י.

תוצאות הסימולציה עם הקוד המקורי בהבדל אחד : בתא 0x64 יש ערך 0x5 :



וכמובן, כאן בסיום התוכנית יש את הערך 3 באוגר t2 – איבר חמישי בסדרת פיבונאצ'י.

תוצאות הסימולציה עם הקוד המקורי בהבדל אחד : בתא 0x64 יש ערך 8- (0xfffffff8) :



סעיף ד

מטלה: הדפיסו והגישו את כל הקוד. סמנו או החלקים בקוד אותם הוספתם/שיניתם. הקוד שהוספתם אמור להכיל הערות מפורטות.

פתרון: בדפים המצורפים נמצא כל הקוד, כאשר השינויים שלנו מודגשים.