אוניברסיטת בר-אילן

הפקולטה להנדסה

BAR-ILAN UNIVERSITY

Engineering Faculty

תכן לוגי ומבוא להנדסת מחשבים

תשע"ה סמס' א' מועד א'

83-253

- מרצה : פרופ' שמואל וימר
- מתרגלים : מר יוסף לונדון, מר עמית מנדלבום
- **חומר עזר מותר:** ספר הקורס בלבד (אפשרי גם מודפס ונקי מכתב יד), שקפי הקורס מודפסים, קובץ פקודות אסמבלי מודפס.
- משך המבחן:

  לסטודנטים משנה ב' הלומדים קורס תכן לוגי לראשונה – שעתיים וחצי.

  לשאר הסטודנטים - שלוש שעות.
- לסטודנטים משנה ב' הלומדים קורס תכן לוגי לראשונה – חובה לענות על שתי השאלות הראשונות בלבד. אין לענות על שאלה 3.
- לשאר הסטודנטים – חובה לענות על כל שלושת השאלות.
- **יש לנמק את כל תשובותיכם**. אין צורך לפתח מחדש תוצאות שהוכחו בכיתה, אלא אם כן נאמר מפורשות לעשות כן.
- יש לשרטט דיאגראמות באופן ברור !
- הכתיבה בעט בלבד. כתיבה בעפרון לא תיבדק.
- יש לכתוב רק בחלק הדף המיועד לכך.
- סה"כ הניקוד הינו 100 (בסוגריים לנבחני שנה ג' וד')

בהצלחה!

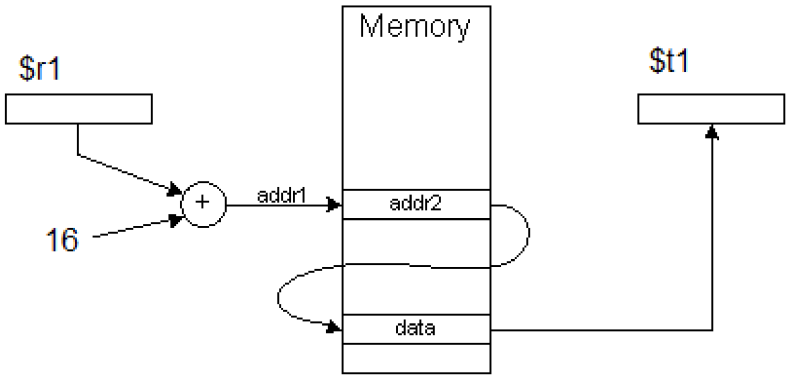
<u>שאלה 1</u> – 60 נק' (44 לתלמידים שאינם בשנה ב')
טעינת מילה מהזכרון בכתובת הרשומה בזכרון.

עליכם לתכנן פקודה חדשה $memlw$, שעבורה OPCODE = 39 , מטרת הפקודה היא לטעון מילה מהזכרון לתוך רגיסטר. מיקומה של המילה הרצויה לטעינה אינו ידוע ישירות, אך מיקומה רשום בתוך הזכרון. מיקום המידע הנ"ל בהיסט (OFFSET) ידוע מכתובת ידועה בזכרון.

להלן הדגמה: $memlw \quad \$t1\,,16(\$r1)$.

הפקודה סוכמת את ההיסט 16 עם הערך שנמצא בתוך רגיסטר $\$r1$.

לאחר מכן ניגשים לזכרון לכתובת הנ"ל וקוראים מילה המשמשת ככתובת שממנה נקרא הערך הנכתב לתוך הרגיסטר $\$t1$.



נדרש לממש את הפקודה ב multicycle mips:

א. האם נדרש שינוי לרצף הנתונים (DATAPATH)? בכדי לענות על השאלה הגדירו את הבעיה ואת תצורת הפתרון.
נראה לפי המימוש שאין ממש צורך בשינוי בDATAPATH אלא רק ב mux שצריך להוסיף לו מצב.
במידה ועקרונית הובן כיצד לממש – הבעיה היתה קריאת מידע ואח"כ שימוש שלה שוב בזכרון.
הפתרון בmulticycle מאוד פשוט מהבחינה הזאת – רק להוסיף מצב נוסף ולחזור לרצף הרגיל של LW

ב. האם נדרש שינוי במערכת הבקרה? במדה וכן, איזה שינוי?
השינויים מובאים בהמשך

ג. עדכנו בהתאם את תרשים מכונת המצבים המצורפת, ורשמו באופן ברור את השינויים שעשיתם וכמו כן גם את קווי הבקרה ושינויי החמרה בסכימה.

ד. ממשו את הבקר ע"י מיקרו פקודות כפי שנלמד בהרצאה, ועדכנו את טבלת המיקרו פקודות בהתאם.

ה. במידה ונדרש לעדכן את הלוגיקה השלטת בזכרון המיקרו פקודות, עדכנו בהתאם (טבלאות ROM וכד').

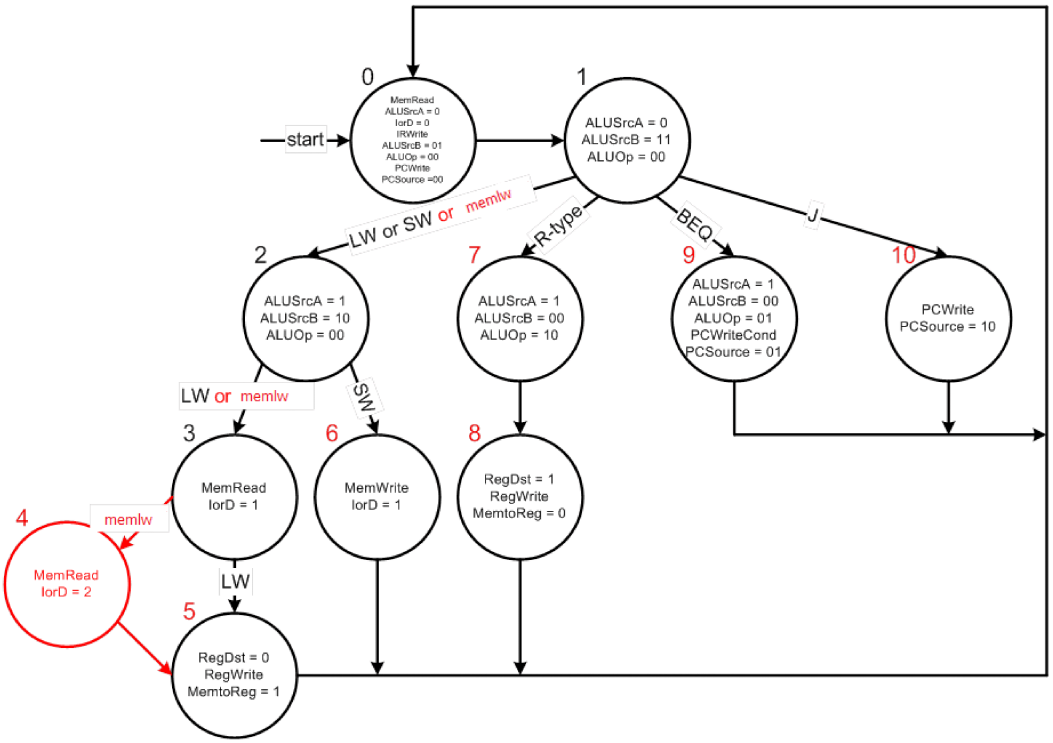
ו. רישמו את כל השינויים הנדרשים בקוד ה VERILOG (מופיע בסוף הבחינה) של מכונת המצבים המממשת את הבקר (מצבים ואותות בקרה).

ז. האם הפקודה ניתנת למימוש בsingle cycle mips ?. נמקו היטב את תשובתכם.

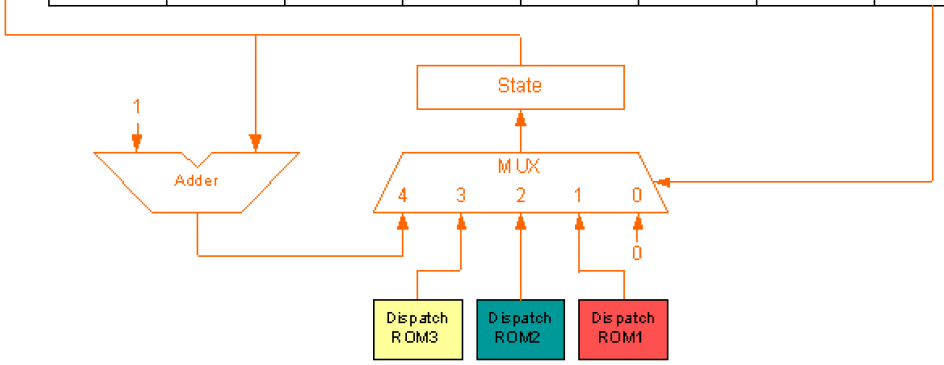
בsingle cycle mips אין את האפשרות לכתוב את המידע שנקרא לרגיסטר ואין את האפשרות לשנות קווי בקרה בזמן של מחזור יחידי. לא ניתן לדגום את המידע ולהכנס שוב עם מידע אחר לתוך הזכרון באותו מחזור שעון

ח. האם הפקודה ניתנת עקרונית למימוש ב PIPELINE? נמקו היטב את תשובתכם.

ב PIPELINE יש לנו את האפשרות להאט את ה PIPELINEכלומר לבצע STALL לפקודות שנכנסו ל PIPE ולחזור על אותו שלב שוב. המידע שנדגם בין שלב לשלב ניתן לשימור ברגיסטרים של ה PIPE ולכן עקרונית נצטרך אומנם להוסיף קוי בקרה וmuxים לפני הזכרון בכדי להחליט את מי להכניס לזכרון אך ניתן ליצור קוי בקרה מתאימים שיבצעו את ה STALLבמקרה הצורך וכן יכניסו את המידע הנדגם מחדש לתוך הpipe



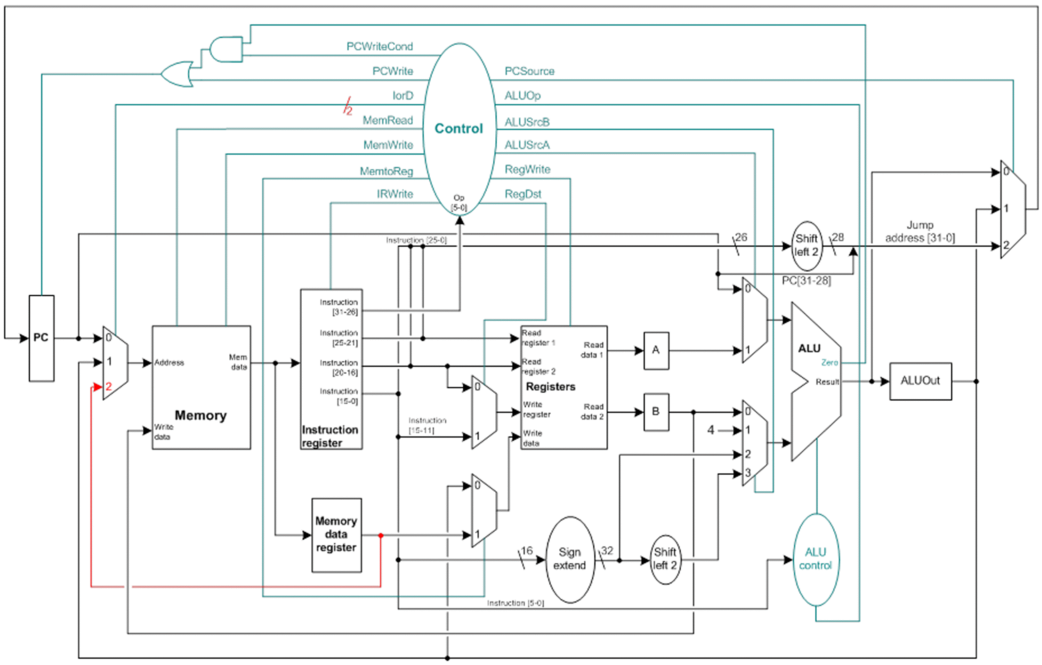
Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch1
Mem1	Add		Extend				Dispatch2
LW2					Read ALU		Dispatch3
memlw					Read MDR		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch
BEQ1	Subt	A	B			ALU Out-cond	Fetch
Jump1						Jump address	Fetch



Dispatch ROM 3		
35	LW	5
39	memlw	4

Dispatch ROM 2		
35	LW	3
43	SW	6
39	memlw	3

Dispatch ROM 1		
0	R-type	7
2	jmp	10
4	beq	9
35	LW	2
43	SW	2
39	memlw	2



<u>שאלה 2</u> 40 נק' (28 נק' לתלמידים שאינם בשנה ב')

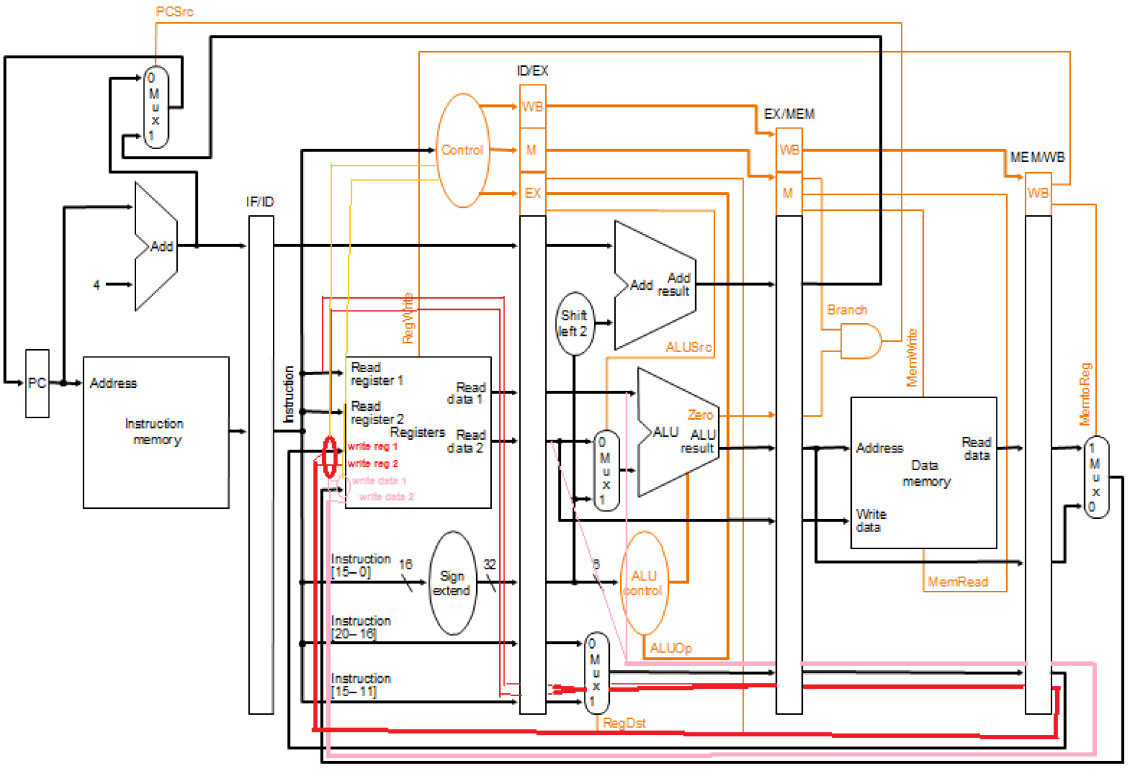
נדרש לממש את הפקודה SWAP המחליפה בין הערכים של שני רגיסטרים בקובץ הרגיסטרים מבלי לשנות את מספר הדרגות בPIPELINE.

א. הצע פורמט, מבין הקיימים, לפקודה על כל המשתמע מכך (אלו אופרנדים תקבל הפקודה וכו').
התשובה יכולה כמובן להיות או RTYPE או ITYPE העיקר לקבל 2 רגיסטרים כאופרנדים. את האופרנד השלישי ניתן לקבוע או כרגיסטר האפס או כקבוע האפס

ב. האם ישנה בעיה בקובץ הרגיסטרים כפי שנלמד בקורס, במידה וכן, מהי הבעיה ומהו הפתרון? במידה ותשובתכם שלילית (אין בעיה), הסבירו כיצד בכוונתכם לממש את הפקודה.
הבעיה המרכזית היא כמובן החוסר ב2 כניסות לכתיבה (ו2 כניסות ליעדי כתיבה) שכן אנחנו מחויבים לכתוב לשני רגיסטרים בכדי לממש את הפקודה

ג. הציעו פתרון חומרה למימוש הפקודה החדשה וממשו אותו.

שרטטו על המעגל של PIPELINE MIPS את התוספות נדרשות (קוי בקרה, בוררים, וכו'). במדה ונדרשים שינויים ברגיסטרי הצינור, ציינו זאת.



ד. האם מנגנון הקידום של DATA HAZARD שנלמד בכתה מסוגל להתמודד עם הפתרון שהצעתם. נמקו בפירוט. (לא נדרש מימוש).

שאלה 3 (28 נק' לתלמידים שאינם בשנה ב')

נתון קוד בASSEMBLY שממש את הלולאה הבאה בC.

```
for (i = 0 ; i < 7 ; i++)
{
        X[i] = A[i] +B[i];
        Y[i] = A[i] – B[i];
}
```

הכתובת ההתחלתית של המערכים A,B,X,Y נמצאים ברגיסטרים s1,s2,s3,s4 בהתאמה. כמו כן, הכתובת שלהם מתחלקת בגודל ה CACHE ללא שארית.

הערך שעבורו הלולאה נעצרת שמורה ברגיסטר t0.

	addi	$t1 , $s1,0	&A[0] -> $t1
	addi	$t2 , $s2, 0	&B[0] -> $t2
	addi	$t3 , $s3, 0	&X[0] -> $t3
	addi	$t4 , $s4, 0	&Y[0] -> $t4
	addi	$t7 , $ZERO,0	0 -> $t7 counter =0
Loop:	lw	$t8, 0 ($t1)	A[i] -> $t8
	lw	$t9, 0 ($t2)	B[i] -> $t9
	add	$t5, $t8 , $t9	A[i]+B[i] = X[i] -> $t5
	sw	$t5, 0 ($t3)	X[i] -> &$t3
	lw	$t9, 0 ($t2)	B[i] -> $t9

	lw	$t8, 0 ($t1)	A[i] -> $t8
	sub	$t6, $t8 , $t9	A[i]-B[i] = Y[i] -> $t6
	sw	$t6, 0 ($t4)	Y[i] -> &$t4
	addi	$t1 , $t1,4	&A[i+1] -> $t1
	addi	$t2 , $t2, 4	&B[i+1] -> $t2
	addi	$t3 , $t3, 4	&X[i+1] -> $t3
	addi	$t4 , $t4, 4	&Y[i+1] -> $t4
	addi	$t7 , $t7, 1	$t7 +1 -> $t7 counter =0
	bne	$t7, $t0 , Loop	COUNTER =? Stop condition

גודל של בלוק בזיכרון הוא 4 מילים .זמן ביצוע כל הפקודות הינו 1ns כולל קריאה וכתיבה לזיכרון במקרה של HIT. המחיר הכולל של MISSהינו 100ns.

נתון שה CACHE הינו DIRECTLY MAPPED וגדלו 32 מלים.

א. מהו זמן ביצוע התוכנית? (הניחו single cycle mips) הסבירו כל שלב ושלב בתשובה.

יש בעיה של גישה מרובה לCACHE. A אחר כך B מוחק אותו אחר כך X וכו... תמיד יש MISS שכן כולם ממופים לאותו מקום.

ב. במקרה וגודל הבלוק משתנה ל2 מילים, האם תשובתכם לסעיף א תשתנה? נמקו.

לא.

ג. במקרה וה associativity של ה CACHE הופכת לWAY-2 עם שגרת החלפה של LRU, האם תשובתכם לסעיף א תשתנה?

הפעם 2 way cache עוזר ב MISS אחד. נראה שני מחזורים. המחזורים הבאים כמחזור השני עם שני HIT ושלושה MISS.

A MISS	A	NAN		A HIT	A	Y
B MISS	A	B		B MISS	A	B
X MISS	X	B		X MISS	X	B
B HIT	X	B		B HIT	X	B
A MISS	A	B		A MISS	A	B
Y MISS	A	Y		Y MISS	A	Y

ד. חיזרו על סעיף ג' עבור 4-WAY?

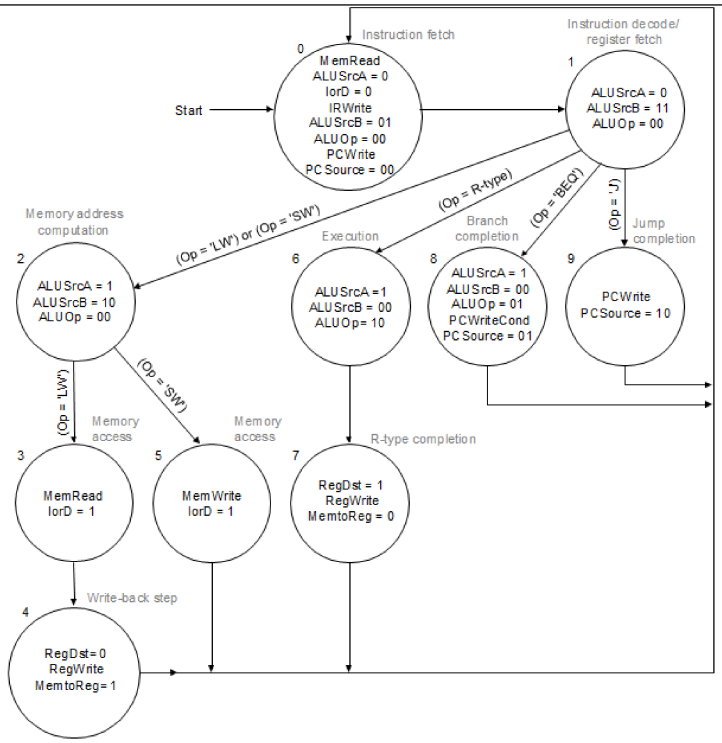
לא יהיו לנו MISS.

ה. מה היה קורה לו סדר השורות הבאות בקוד היו מוחלפות האחת בשניה:

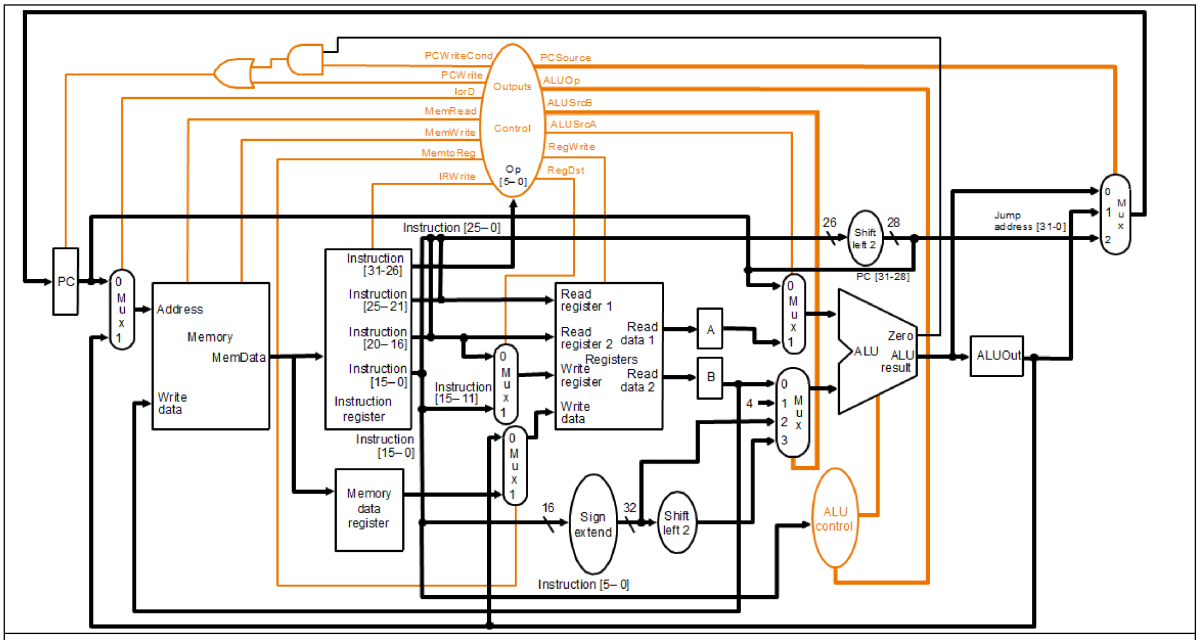
	lw	$t9, 0 ($t2)
	lw	$t8, 0 ($t1)

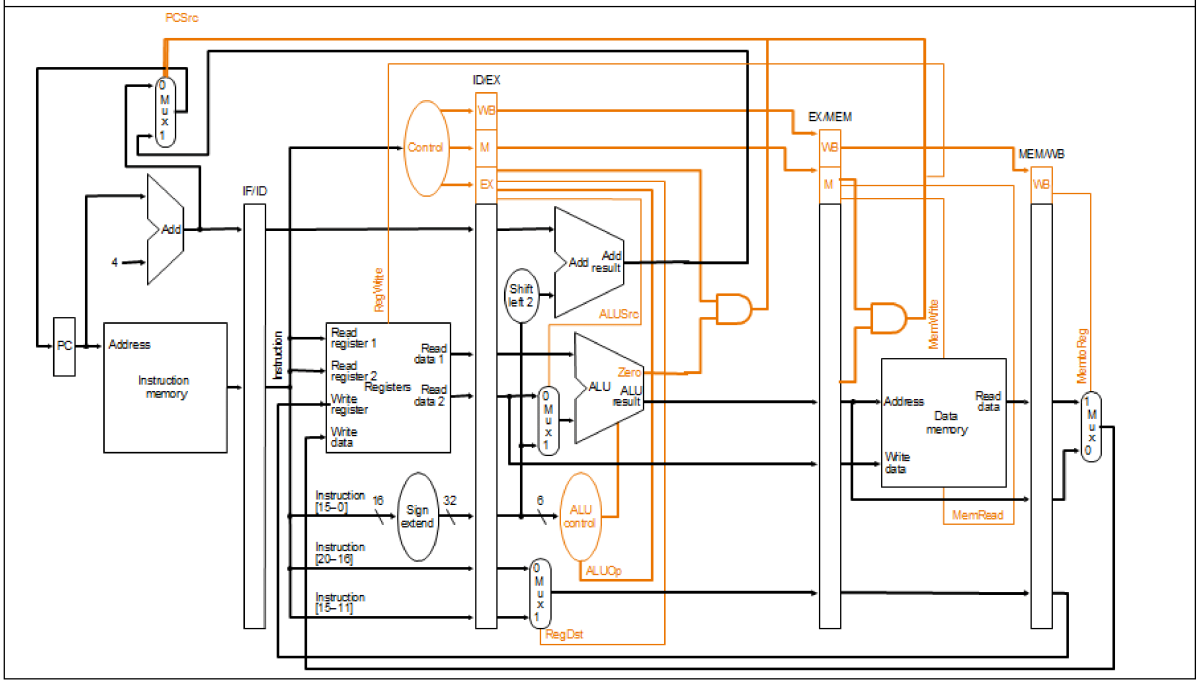
כפי שניתן לראות נצבור MISS תמיד בway2 וב4 או יחיד לא ישנה כלום:

A MISS	A	NAN		A MISS	A	Y
B MISS	A	B		B MISS	A	B
X MISS	X	B		X MISS	X	B
A MISS	X	A		A MISS	X	A
B MISS	B	A		B MISS	B	A
Y MISS	B	Y		Y MISS	B	Y



STATE	Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
0	Fetch	Add	PC	4		Read PC	ALU	Seq
1		Add	PC	Extshft	Read			Dispatch 1
2	Mem1	Add	A	Extend				Dispatch 2
3	LW2					Read ALU		Seq
4					Write MDR			Fetch
5	SW2					Write ALU		Fetch
6	Rformat1	Func code	A	B				Seq
7					Write ALU			Fetch
8	BEQ1	Subt	A	B			ALUOut-cond	Fetch
9	JUMP1						Jump address	Fetch


PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op [5–0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
Instruction [25–0]
26
Shift left 2
28
PC [31-28]
Jump address [31-0]
PC
Address
Memory
MemData
Write data
Instruction [31-26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–0]
Instruction register
Instruction [15–0]
Instruction [15–11]
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
A
B
4
ALU
Zero
ALU result
ALUOut
Memory data register
16
Sign extend
32
Shift left 2
ALU control
Instruction [5–0]


PCSrc
ID/EX
EX/MEM
MEM/WB
WB
M
EX
Control
IF/ID
Add
4
PC
Address
Instruction memory
Instruction
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALUSrc
Zero
ALU
ALU result
Instruction [15–0]
16
Sign extend
32
6
ALU control
ALUOp
Instruction [20–16]
Instruction [15–11]
RegDst
MemWrite
MemRead
Address
Data memory
Read data
Write data
MemtoReg

ב. לפניכם מכונת המצבים של Multi-Cycle MIPS הממומשת בשפת Verilog ב. לפניכם מכונת המצבים של Multi-Cycle MIPS הממומשת בשפת Verilog, במידה והוספתם מצבים, אנא רשמו את השינויים והתוספות הנדרשים בקוד, שימו לב לשינויים, אם ישנם, בקווי הבקרה ורשמו גם אותם.
שימו לב: ה-Opcode של הפקודה החדשה הינו 39.

```
1   //This is the state machine of the multi-cycle MIPS controller
2
3   module multiFSM(clk, rstn, OpCode, MemRead, MemWrite, ALUSrcA,   ALUSrcB, IorD,
4                               ALUOp, PCWrite, PCSource, PCWriteCond, RegWrit  e, RegDst
5                                   MemtoReg, IRWrite);
6
7   //Signal interface
8   input clk;
9   input rstn;
10   input [5:0] OpCode;
11
12   output reg MemRead;
13   output reg MemWrite;
14   output reg ALUSrcA;
15   output reg [1:0] ALUSrcB;
16   output reg IorD;
17   output reg [1:0] ALUOp;
18   output reg PCWrite;
19   output reg [1:0] PCSource;
20   output reg PCWriteCond;
21   output reg RegWrite;
22   output reg RegDst;
23   output reg MemtoReg;
24   output reg IRWrite;
25
26   //Signals for current and next State
27   reg [3:0] current_state;
28   reg [3:0] next_state;
29
30   //Current state register
31   always @(posedge clk or negedge rstn)
32     if(~rstn)
33       current_state <= STATE_0;
34     else
35       current_state <= next_state;
36
37   //The state machine
38   always @(current_state, OpCode)
39     begin
40       case(current_state)
41         STATE_0: //Fetch
42                             begin
43                               next_state = STATE_1;
44                               MemRead = 1'b1;
45                               ALUSrcA = 1'b1;
46                               IorD = 1'b1;
47                               IRWrite = 1'b1;
48                               ALUSrcB = 2'b01;
49                               ALUOp = 2'b00;
50                               PCWrite = 1'b1;
```

```
51                      PCSource = 2'b00;
52                    end
53      STATE_1:
54                    begin
55                      //Addressing dispatch ROM 1
56                      case(OpCode)
57                              5'd0 : next_state = STATE_6;
58                              5'd2 : next_state = STATE_9;
59                              5'd4 : next_state = STATE_8;
60                              5'd35: next_state = STATE_2;
61                              5'd43: next_state = STATE_2;
62                      endcase
63                      ALUSrcA = 1'b0;
64                      ALUSrcB = 2'b11;
65                      ALUOp = 2'b00;
66                    end
67      STATE_2: //Mem1
68                    begin
69                      //Addressing dispatch ROM 2
70                      case(OpCode)
71                              5'd35 : next_state = STATE_3;
72                              5'd43 : next_state = STATE_5;
73                      endcase
74                      ALUSrcA = 2'b1;
75                      ALUSrcB = 2'b10;
76                      ALUOp = 2'b00;
77                    end
78      STATE_3: //LW2
79                    begin
80                      next_state = STATE_4;
81                      MemRead = 1'b1;
82                      IorD = 1'b1;
83                    end
84      STATE_4:
85                    begin
86                      next_state = STATE_0;
87                      RegDst = 1'b0;
88                      RegWrite = 1'b1;
89                      MemtoReg = 1'b1;
90                    end
91      STATE_5: //SW2
92                    begin
93                      next_state = STATE_0;
94                      MemWrite = 1'b1;
95                      IorD = 1'b1;
96                    end
97      STATE_6: //Rformat 1
98                    begin
99                      next_state = STATE_7;
100                     ALUSrcA = 1'b1;
101                     ALUSrcB = 2'b00;
102                     ALUOp = 2'b10;
103                   end
104     STATE_7:
105                   begin
106                     next_state = STATE_0;
```

```
107                         RegDst = 1'b1;
108                         RegWrite = 1'b1;
109                         MemtoReg = 1'b0;
110                       end
111       STATE_8: //BEQ1
112                       begin
113                         next_state = STATE_0;
114                         ALUSrcA = 1'b1;
115                         ALUSrcB = 2'b00;
116                         ALUOp = 2'b01;
117                         PCWriteCond = 1'b1;
118                         PCSource = 2'b01;
119                       end
120       STATE_9: //JUMP1
121                       begin
122                         next_state = STATE_0;
123                         PCWrite = 1'b1;
124                         PCSource = 2'b10;
125                       end
126     endcase   //end case(current_state)
127  end  //end state machine
128
129  endmodule
```

<u>פתרון הסעיף</u>

```
//line 16 change
output [1:0] IorD

//line 46 change
IorD = 2'b01;

//line 60 add
5'd39: next_state = STATE_2;

//line 71 add
5'd39: next_state = STATE_3;

//line 80 change
case(OpCode)
        5'd35: next_state = STATE_4;
        5'd39: next_state = STATE_10;
endcase

//line 82 change - אפשר לא להוריד אם שוכחים את זה
IorD = 2'b01;

//line 95 change - אפשר לא להוריד אם שוכחים את זה
IorD = 2'b01;

//line 125 add
STATE_10:
        begin
```

```
next_state = STATE_4;
MemRead = 1'b1;
IorD = 2'b10;
```