|  |  |  |
| --- | --- | --- |
| **BAR-ILAN UNIVERSITY** Engineering Faculty |  | אוניברסיטת בר-אילן הפקולטה להנדסה |

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!**

שאלה 1 **– 60 נק' (44 לתלמידים שאינם בשנה ב')**

**טעינת מילה מהזכרון בכתובת הרשומה בזכרון.**

עליכם לתכנן פקודה חדשה $memlw$, שעבורה 39 =OPCODE , 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: $memlw \$t1 , 16(\$r1)$.

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:

1. 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 בmux 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
2. האם נדרש שינוי במערכת הבקרה? במדה וכן, איזה שינוי?
השינויים מובאים בהמשך
3. עדכנו בהתאם את תרשים מכונת המצבים המצורפת, ורשמו באופן ברור את השינויים שעשיתם וכמו כן גם את קווי הבקרה ושינויי החמרה בסכימה.
4. ממשו את הבקר ע"י מיקרו פקודות כפי שנלמד בהרצאה, ועדכנו את טבלת המיקרו פקודות בהתאם.
5. במידה ונדרש לעדכן את הלוגיקה השלטת בזכרון המיקרו פקודות, עדכנו בהתאם (טבלאות ROM וכד').
6. רישמו את כל השינויים הנדרשים בקוד ה VERILOG (מופיע בסוף הבחינה) של מכונת המצבים המממשת את הבקר (מצבים ואותות בקרה).
7. 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

1. האם הפקודה ניתנת עקרונית למימוש ב PIPELINE? נמקו היטב את תשובתכם.
בPIPELINE יש לנו את האפשרות להאט את הPIPELINE כלומר לבצע STALL לפקודות שנכנסו לPIPE ולחזור על אותו שלב שוב. המידע שנדגם בין שלב לשלב ניתן לשימור ברגיסטרים של הPIPE ולכן עקרונית נצטרך אומנם להוסיף קוי בקרה וmuxים לפני הזכרון בכדי להחליט את מי להכניס לזכרון אך ניתן ליצור קוי בקרה מתאימים שיבצעו את הSTALL במקרה הצורך וכן יכניסו את המידע הנדגם מחדש לתוך הpipe





שאלה 2 **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.

1. 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
2. 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
3. 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.



1. 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 הCACHE 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 MISS הינו 100ns.

נתון שה CACHE הינו DIRECTLY MAPPED וגדלו 32 מלים.

1. 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.
2. במקרה וגודל הבלוק משתנה ל2 מילים, האם תשובתכם לסעיף א תשתנה? נמקו.
לא.
3. במקרה והassociativity של הCACHE הופכת ל2-WAY עם שגרת החלפה של LRU, האם תשובתכם לסעיף א תשתנה?
הפעם 2 way cache עוזר בMISS אחד. נראה שני מחזורים. המחזורים הבאים כמחזור השני עם שני HIT ושלושה MISS.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Y | A | A HIT |  | NAN | A | A MISS |
| B | A | B MISS |  | B | A | B MISS |
| B | X | X MISS |  | B | X | X MISS |
| B | X | B HIT |  | B | X | B HIT |
| B | A | A MISS |  | B | A | A MISS |
| Y | A | Y MISS |  | Y | A | Y MISS |

1. חיזרו על סעיף ג' עבור 4-WAY?
לא יהיו לנו MISS.
2. 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 ב2way וב4 או יחיד לא ישנה כלום:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Y | A | A MISS |  | NAN | A | A MISS |
| B | A | B MISS |  | B | A | B MISS |
| B | X | X MISS |  | B | X | X MISS |
| A | X | A MISS |  | A | X | A MISS |
| A | B | B MISS |  | A | B | B MISS |
| Y | B | Y MISS |  | Y | B | Y MISS |

|  |
| --- |
|  |
|  |
|  |
|  |

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, lorD,

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 lorD;

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 lorD = 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 lorD = 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 lorD = 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

**פתרון הסעיף**

**//line 16 change**

output [1:0] lorD

**//line 46 change**

lorD = 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**

lorD = 2'b01;

**//line 95 change - אפשר לא להוריד אם שוכחים את זה**

lorD = 2'b01;

**//line 125 add**

STATE\_10:

 begin

 next\_state = STATE\_4;

 MemRead = 1'b1;

 lorD = 2'b10;