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

תכן לוגי ומבוא למחשבים

# תשע"ו סמס' א' מועד ב'

**83-253**

* מרצה : פרופ' שמואל וימר
* מתרגלים : מר יוסף לונדון, מר איתמר לוי
* **חומר עזר מותר:** שקפי ההרצאות כולל הערות והסברים שנרשמו במהלך ההרצאות וספר הקורס.
* משך המבחן:

שלוש שעות.

* חובה לענות על כל השאלון. סה"כ נקודות אפשרי 110, הציון הסופי לא יעלה בכל מקרה על 100.
* **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.

**בהצלחה!**

שאלה 1**(40 נקודות)**

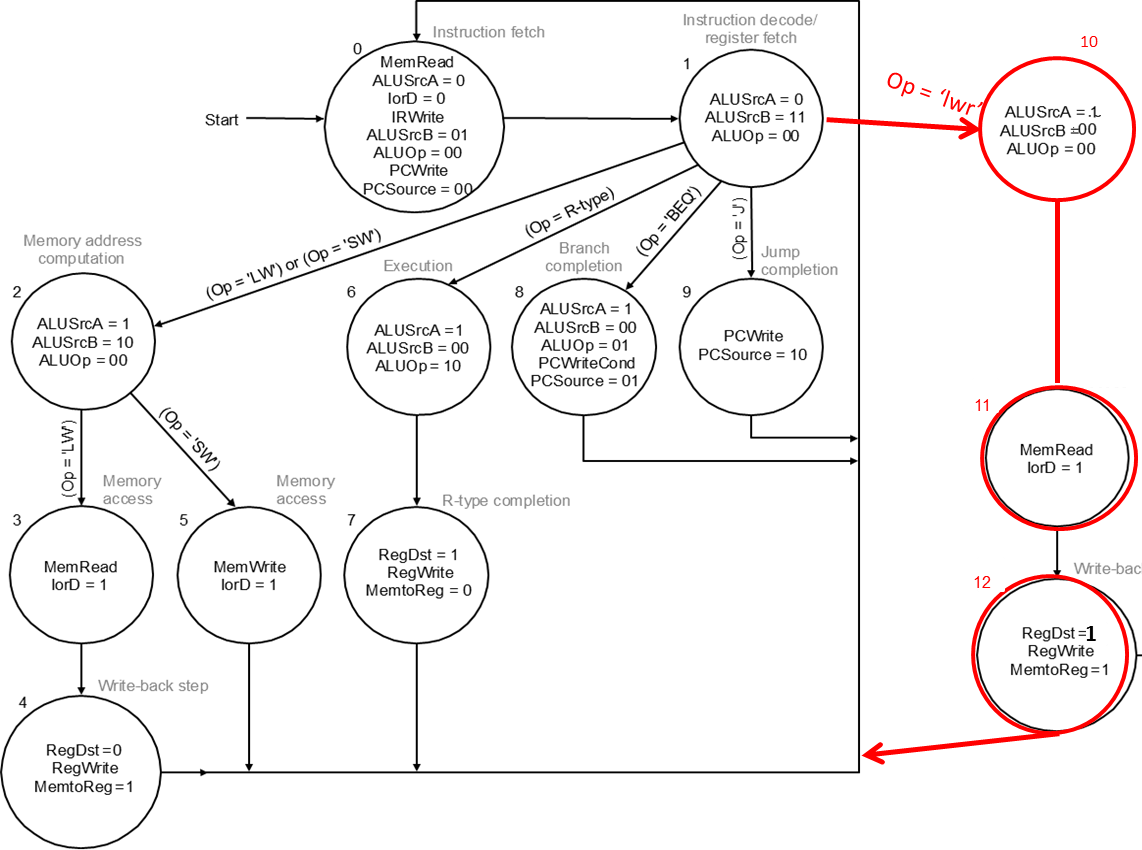
בכיתה תרגלנו את הפקודה lwr אשר מקבלת שלושה רגיסטרים כאופרנדים למערכת single cycle mips. הפקודה סוכמת את אשר בשניים מהם וניגשת לזיכרון ומוציאה מילה מכתובת הסכום. המילה המוצאת נכתבת לתוך הרגיסטר השלישי. Lwr $R1 , $R2, $R3

1. ממשו את הפקודה במלואה למערכת multi cycle mips.

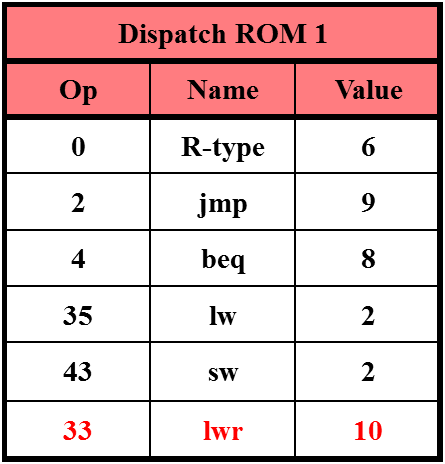
הכוונה במלואה היא שינוי הDATA PATH במידת הצורך ועדכון הבקר.

הצהירו על מבנה פקודה מתאים, תאמו את כל הפרטים הנצרכים לפתרון. עדכנו את כל קוי הבקרה שצריכים עדכון בכל שלב. מלאו מחדש את דיאגרמת המצבים לפי הצורך. עדכנו את טבלאות הROM במידה וצריך ואת מערכת השליטה של הבקר.

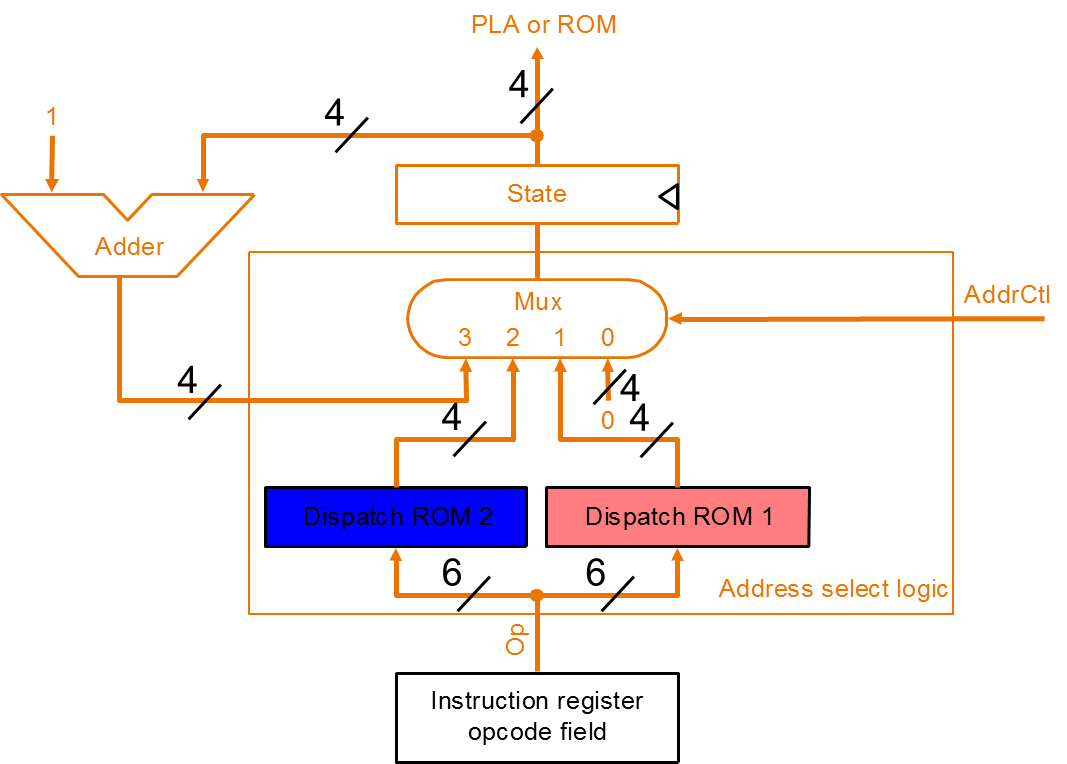
במידת האפשר השתמשו במצבים קיימים מראש.

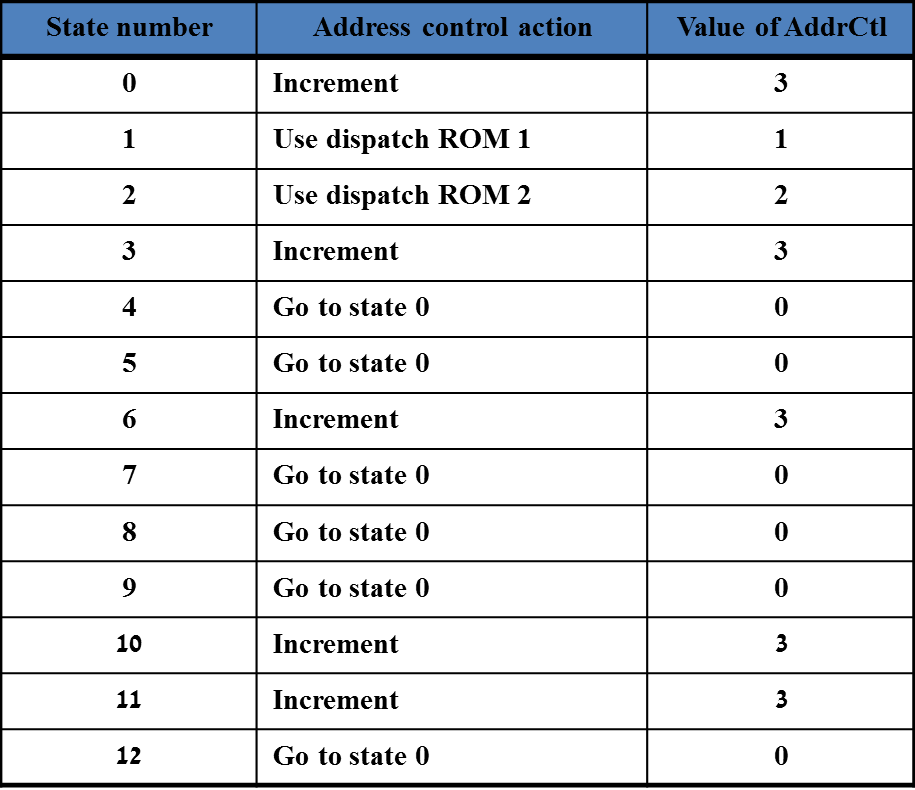


אין מצב אשר מבצע סכימה של שני רגיסטר לפקודה שאיננה RTPYPE מבחינת הOPCODE שלה. RTYPE רגיל לא יכול לגשת לזכרון או ליתר דיוק, המערכת לא תדע לזהות אם זה לא RTYPE במידה והOPCODE יהיה 000000. לכן היה צורך במצב נוסף. השלמת שאר המצבים היתה פשוטה יותר מבחינת מערכת שליטת המצבים. ניתן היה לקפוץ למצב 3 ומשם להתפצל לחדש שכן צריך לשנות את הקו בקרה של REGDST.

 עדכון טבלת הROM ממצב 1.

אין עדכון בבקר שינוי המצבים.





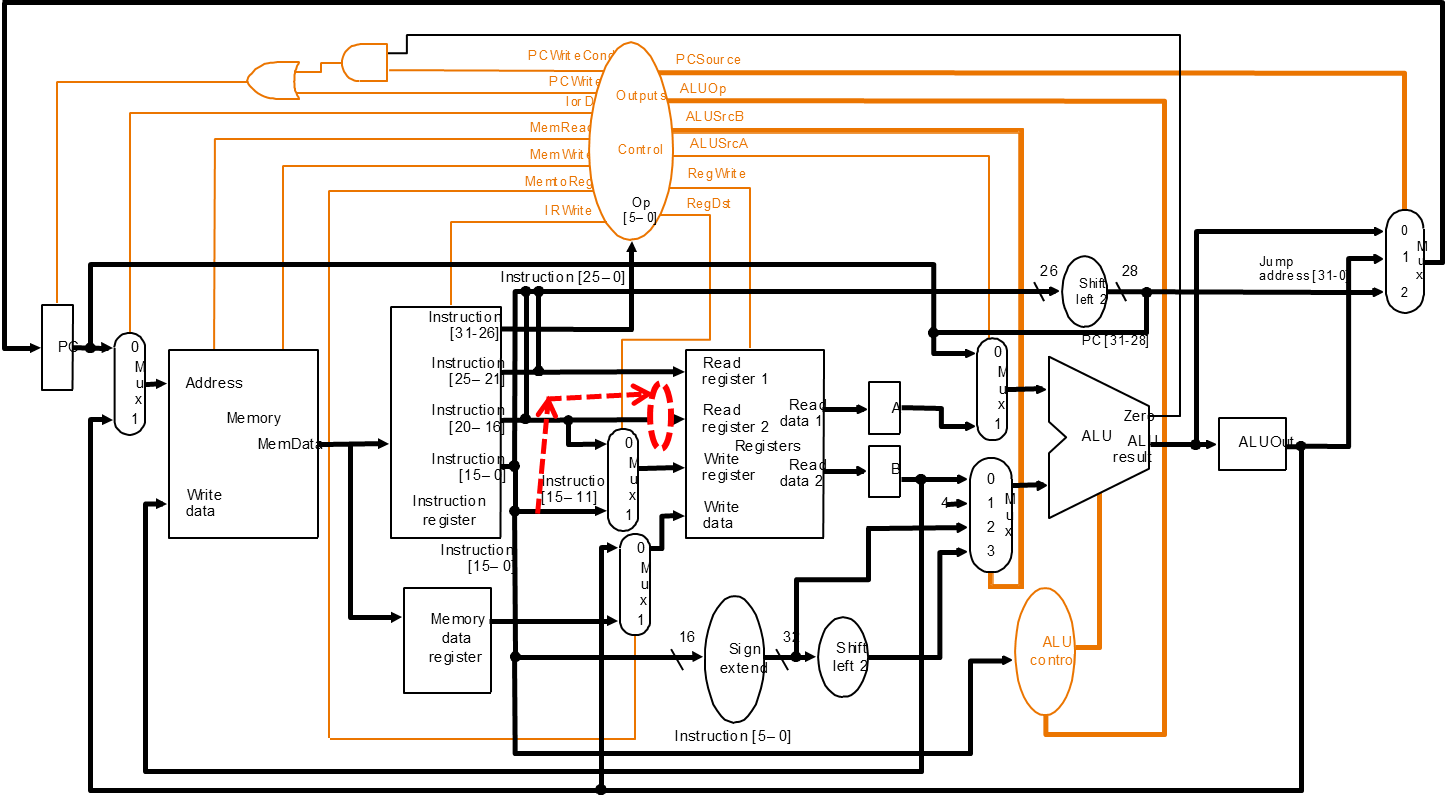
ועדכון מערך השליטה.

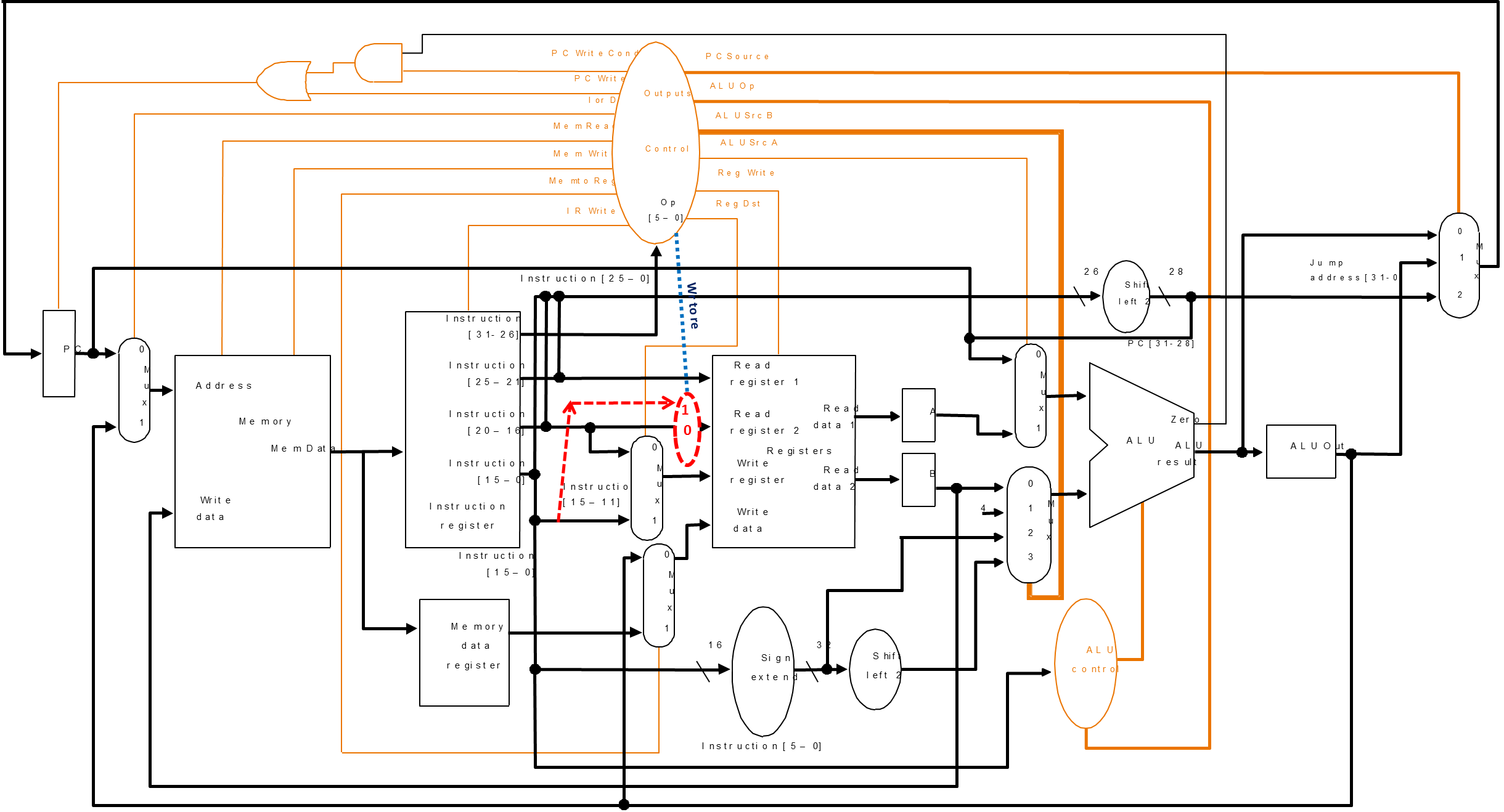
Swr $R1 , $R2, $R3

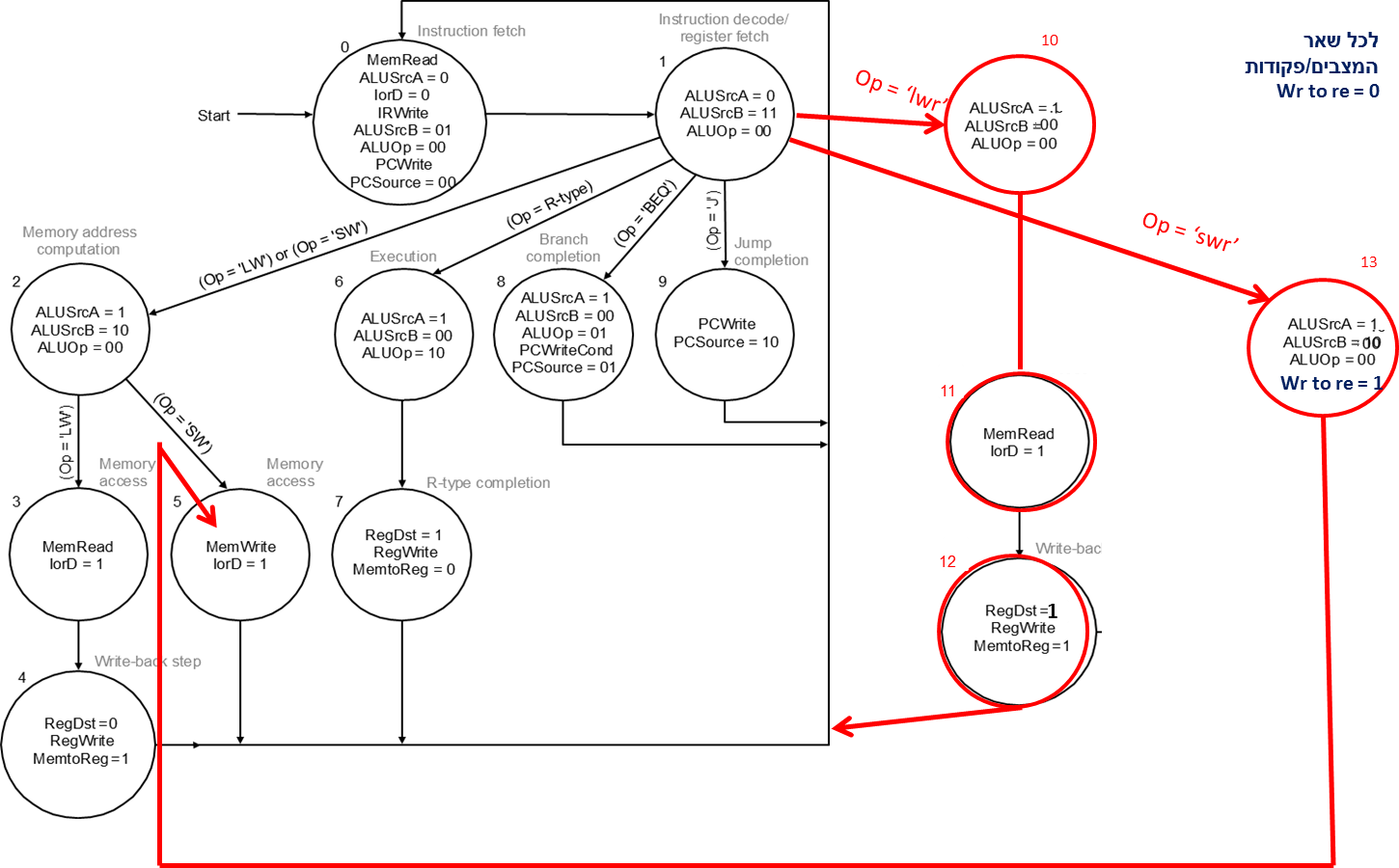
Swr הינה פקודה חדשה אשר מבצעת חיבור בין שני רגיסטרים ומכניסה מילה מתוכן רגיסטר שלישי לזיכרון בכתובת של החיבור. (כיוון הפוך לLWR).

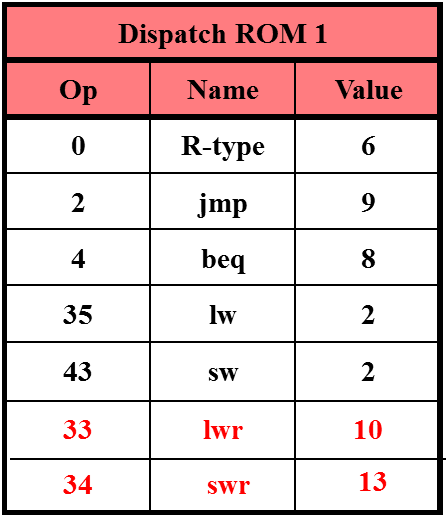
1. האם הפקודה ניתנת למימוש במערכת single cycle mips ללא שינויים בDATAPATH? הסברו פרטו ונמקו.
2. ממשו את הפקודה במערכת multi cycle mips עם אותם דגשים מסעיף א על המערכת מסעיף א. האם היה צורך בעדכון הDATAPATH? במידת האפשר מזערו את השינויים ביחידות מובנות של הDATAPATH (כגון הזיכרון או קבוץ הרגיסטרים) על חשבון שינויים אחרים וקוי בקרה נוספים. עדיף להוסיף בוררים מאשר שינויים משמעותיים.

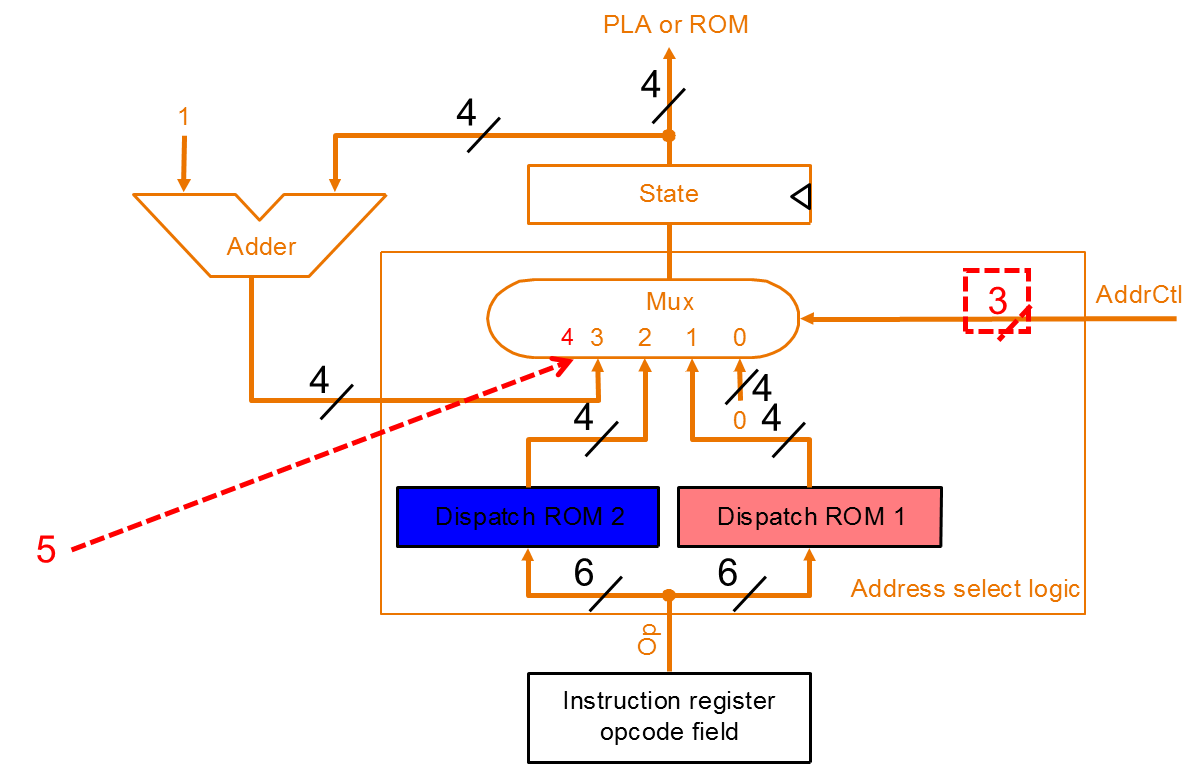
כמעט אותו דבר, צריך להוסיף מצב כדי שיקרא מקובץ הרגיסטרים את הרגיסטר לכתיבה לזכרון בנוסף לחיבור שנעשה באותו זמן בALU. לצורך הקריאה מאופרנד שלישי נצטרך קו בקרה נוסף וmux בכניסה לפורט ה2 של הרגיסטר FILE.

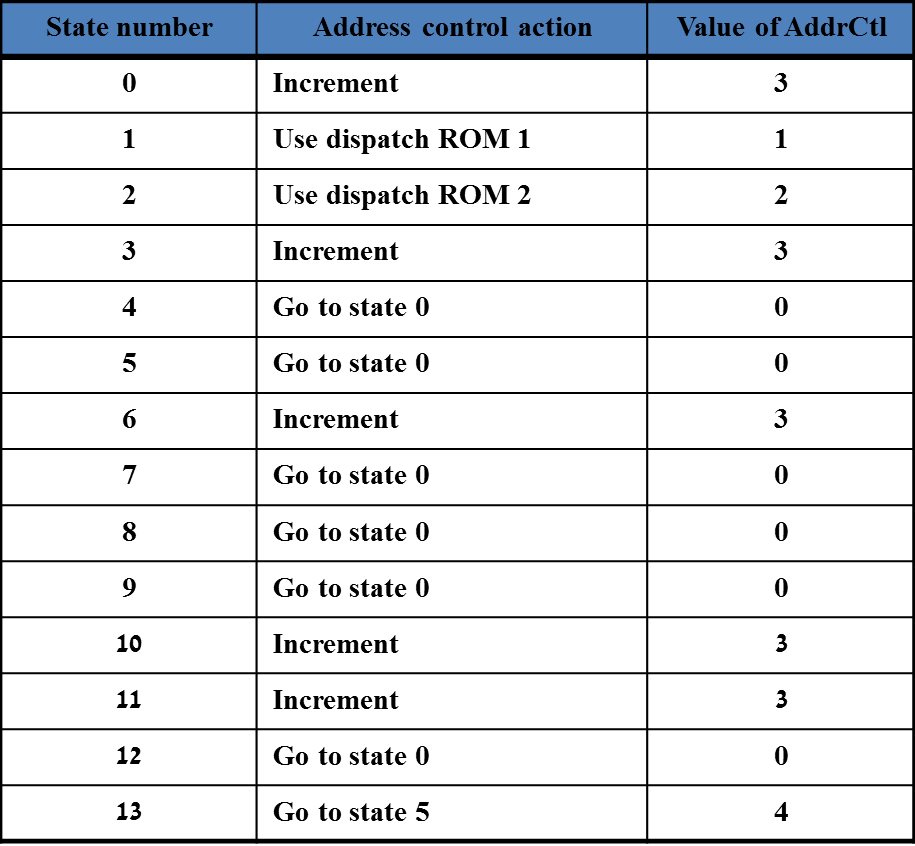












שאלה 2  **(40 נקודות)**

**נדרש לחשב סכום של שני ווקטורים ו ולרשום את התוצאה לווקטור . . נתון שהווקטור נמצא בזיכרון בכתובת** 0x40000040 **ולאחריו מיד נמצא הווקטור . את התוצאה יש לרשום בזיכרון בכתובת מיד לאחר הווקטור . בסיום החישוב יש להפסיק את התכנית ע"י פקודת** halt**. שימו לב שפקודת** halt **מתרחשת בשלה ה** ID**.**

1. **יש לרשום בטבלה המצורפת קוד** assembly **המתאים למעבד המתואר בתרשים המצורף. הקפידו לרשום את הקוד היעיל ביותר.**

**ראשית, יש להניח שהכתובות**  **נמצאות בתוך רגיסטר (לא נאמר מפורשות בשאלה). שימו לב שאין דרך להכנסת כתובת זאת ישירות לתוך רגיסטר ע"י פקודת** addi, **וזאת משום שהכתובות המדוברות הינן יותר מ 16 סיביות! כמו כן, אי אפשר להשתמש בה כ** offset **לפקודת** lw **או** sw, **שוב מאותה סיבה שמדובר ביותר מ 16 סיביות. בכל מקרה, גם אם הכתובות חושבו כמתואר, לא הורדו נקודות.**

**למרות שנדרשות שתי פקודות** lw **ולאחריהן פקודת** add**, ניתן למנוע** load hazard **(כלומר את בזבוז המחזור הנגרם ע"י** hazard detection unit**) ע"י הכנסת פקודה אחרת ביניהן.**

**הקוד הרשום להלן איננו** assembly**, אבל ניתן למיפוי 1:1.**

|  |  |
| --- | --- |
| $t0 = 0x40000040 // base address |  |
| $t1 = 16 // initialize loop index |  |
|  |  |
| $t1 = $t1 – 1 // update loop index | loop: |
| load 0($t0) from mem into $t2 // |  |
| load 64($t0) from mem into $t3 // |  |
| $t0 = $t0 + 4 // update base address |  |
| $t4 = $t3 + $t2 // |  |
| store $t4 in mem at 128($t0) // |  |
| bne $t1 $zero loop |  |
| halt |  |
|  |  |
|  |  |

1. **כמה מחזורי שעון ידרשו לביצוע התכנית שרשמתם?**

**בממוש הנ"ל** load hazard **איננו קורה, לכן ה** hazard detection unit **איננו נכנס לפעולה ואין אבוד מחזורי שעון. בסה"כ ישנן** 2 **פקודות לפני הלולאה,** 7 **פקודות בתוך הלולאה ופקודת העצירה שנקראת גם היא בכל חזרה של הלולאה (אבל מבוטלת ע"י** IF.Flush**). סה"כ** 2+16(7+1)=110.

1. **מהנדס צעיר ונמרץ בוגר בר-אילן, החליט לבטל את הקו** IF.Flush **הנראה בשרטוט. האם הקוד שרשמתם בסעיף א' יפעל באופן תקין? נמקו היטב את תשובתכם.**

**הקוד אינו תקין. התכנית תפסיק לפעול מיד בתום הלולאה הראשונה משום שהפקודה** halt **שנכנסה לצינור תפעל מיד.**

**יש לשים לב שביטול** IF.Flush **אינו משפיע על** ,hazard detection unit**ועלכן במדה ובקוד היו מצבי** load hazard**, הללו ממשיכים להיות מטופלים.**

1. **במדה ותשובתכם הייתה שלילית, שנו את הקוד שרשמתם בסעיף א' כך שהתכנית תפעל נכון, ובנוסף גם באופן היעיל ביותר. הסבירו היטב את השינויים שנעשו.**

**יש להכניס** delay slot **מיד לאחר פקודת הקפיצה המותנית. אפשר כמובן להכניס** nop**, אולם על מנת להימנע מאבדן מחזור שעון, אין צורך ב** nop**וניתן להעביר את הפקודה** store $t4 **אחרי הקפיצה המותנית. לשים לב שפקודת ה** halt **מתבצעת כאשר ה**store **האחרון נמצא בשלב ה** EXE**, ועלכן נדרשת הפרדה ע"י** nop**.**

|  |  |
| --- | --- |
| $t0 = 0x40000040 // base address |  |
| $t1 = 16 // initialize loop index |  |
|  |  |
| $t1 = $t1 – 1 // update loop index | loop: |
| load 0($t0) from mem into $t2 // |  |
| load 64($t0) from mem into $t3 // |  |
| $t0 = $t0 + 4 // update base address |  |
| $t4 = $t3 + $t2 // |  |
| bne $t1 $zero loop |  |
| store $t4 in mem at 128($t0) // |  |
| nop |  |
| halt |  |
|  |  |

1. **המהנדס בוגר בר-אילן הגדיל ראש עוד יותר, והחליט בנוסף לבטל גם את מנגנון** hazard detection unit**. האם הקוד שנרשם בסעיף א' (במדה ולא נדרש שינוי) או הקוד שנרשם ב ד' (במדה ונדרש שינוי) יפעלו באופן תקין? נמקו היטב את תשובתכם. במדה ונדרש שינוי רשמו קוד מתוקן.**

**הקוד הרשום לעי"ל עבוד באופן תקין. תפקיד** hazard detection unit **הינו לטפל ב** load hazards**. באופן בו רשום הקוד המרחק בין פקודות ה** load **לבין הפקודה** add **מספיק גדול, כך שלא תיוצר כל בעיה. במדה והפקודה לחבור** $t4 = $t3 + $t2 היתה **עוקבת מייד אחרי** load 64($t0), **יש להפרידן ע"י** nop.

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

1. **עבור המעבד מסעיף ה' נדרש לחשב את הפעולה . את הערך יש לרשום בזיכרון בכתובת מיד לאחר הווקטור . בסיום החישוב יש להפסיק את התכנית ע"י פקודת** halt**. שימו לב שה** ALU **איננו תומך בפעולת כפל. יש לרשום בטבלה המצורפת קוד** assembly **מתאים. הקפידו לרשום את הקוד היעיל ביותר.**

שימו לב לטעינת האיבר באחרון של הווקטור C למשתנה D המוחזק באוגר $t1.

את פקודת הכפל ניתן לממש ע"י הזזה שמאלה של ארבע סיביות. מאחר והמנגנון hazard detection unit איננו קיים, הטיפול ב load hazard חייב להיעשות ע"י הכנסת nop אחרי load. ה nop השני תפקידו לדאוג שפקודת ה store תסיים את שלב ה MEM ולא תסתיים קודם לכן כתוצאה מ halt.

קוד פחות יעיל יכול להפעיל לולאה 15 פעמים שבה מוסיפים את לעצמו.

אפשרי גם ע"י הכפלה ב 2 בכל סבוב שבו מוסיפים את הצבירה עד עכשיו לעצמה. זה דורש 4 חזרות של הלולאה.

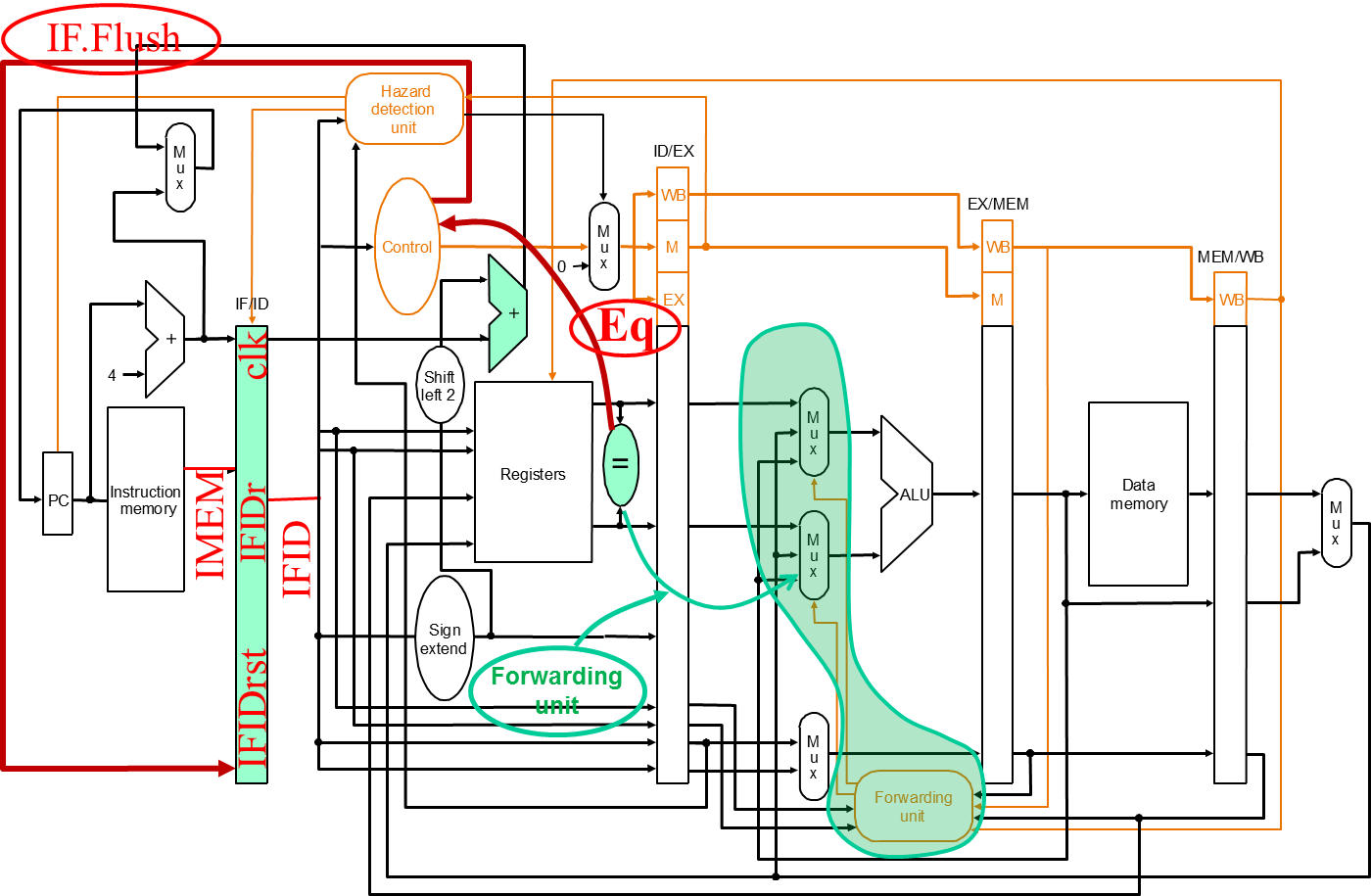
במקרה של שני הפתרונות האחרונים יש לשים לב שמדובר במעבד שבו בוטל IF.Flush, **ולכן יש להרחיק את פקודת**halt **מבדיקת תנאי סיום הלולאה ע"י** delay slot**, אחרת** halt **יבוצע מייד בתום הסבוב הראשון.**

|  |  |
| --- | --- |
| load 188($t0) to $t1 // |  |
| nop |  |
| $t1 = $t1 << 4 // X16 multiply by left shift |  |
| store $t1 in 192($t0) // |  |
| nop |  |
| halt |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

שאלה 3  **(30 נקודות)**

שאלה זו עוסקת בייצור הסיגנל IF.Flush שנידון בשאלה 2 ומודגש בסכמה הבאה.

1. רשום תכן בשפת Verilog שמייצר סיגנל זה (בתוך מודול הMIPS ולא כמודול עצמאי). השתמש בתבנית הניתנת למטה ורשום הקוד באזורים המוקצים לכך באפור. שים לב שאתה מקבל מבנה הצהרות כללי (header) בשפת Verilog של המעבד והצהרות המתאימות לסיגנלים הדרושים לייצר את IF.Flush ועליך להשתלב בהצהרות אלו (כולל הצהרה של רגיסטר IFIDreg). ניתן להשתמש אך ורק בשמות שניתנו על הסכימה (שמוצהרים, או סיגנלים שתצהירו עליהם ותייצרו אותם מהסיגנלים הקיימים). ניקוד ירד על חוסר סדר וחוסר בהסבר\הערות במבנה Verilog \\



module IDIFreg (input reset, clk input [31:0] in\_word, output reg [31:0] out\_word);

always @(posedge clk)

if (reset(

out\_word <= 0;

else if (clk == 1)

out\_word <= in\_word;

endmodule

module MIPS (input clk);

wire clk;

reg[31:0] PC, RegFile[0:31], IMemory[0:1023], DMemory[0:1023], //memories

IDEXA, IDEXB, IDEXIR, EXMEMB, EXMEMALUout, EXMEMIR, MEMWBIR;

wire [31:0] IFID, IMEM;

wire [4:0] IFIDrt, IFIDrs, IFIDrd, IFIDop, IFIDfun;

//Definitions

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

assign IFIDrt=IFID[15:11]; assign IFIDrd=IFID[20:16]; assign IFIDrs=IFID[25:21]; assign IFIDop=IFID[31:26];

…

…

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

endmodule;