אוניברסיטת בר-אילן

הפקולטה להנדסה

BAR-ILAN UNIVERSITY

Engineering Faculty

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!

<u>שאלה 1</u>(40 נקודות)

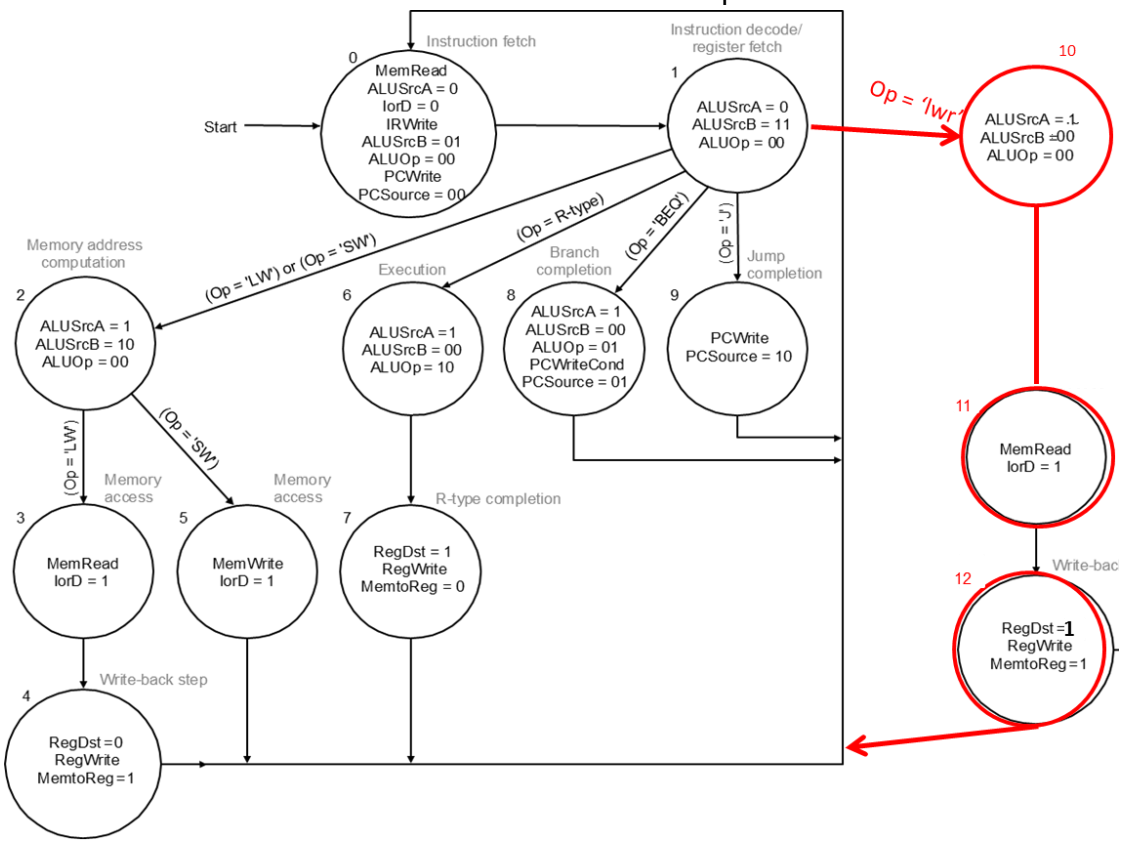
בכיתה תרגלנו את הפקודה lwr אשר מקבלת שלושה רגיסטרים כאופרנדים למערכת single cycle mips. הפקודה סוכמת את אשר בשניים מהם וניגשת לזיכרון ומוציאה מילה מכתובת הסכום. המילה המוצאת נכתבת לתוך הרגיסטר השלישי. Lwr $R1 , $R2, $R3

א. ממשו את הפקודה במלואה למערכת multi cycle mips.

הכוונה במלואה היא שינוי ה DATA 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 ה ROMבמידה וצריך ואת מערכת השליטה של הבקר.

במידת האפשר השתמשו במצבים קיימים מראש.

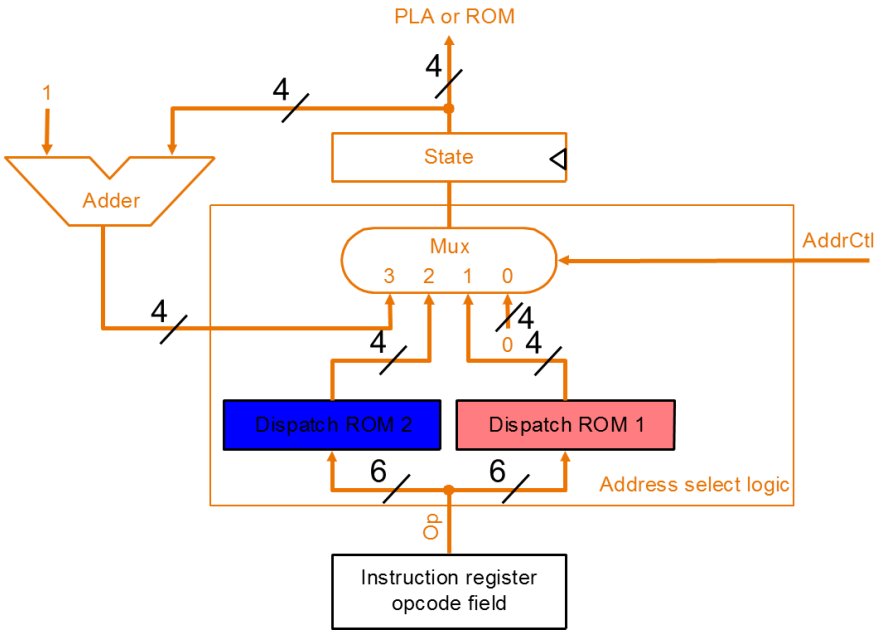


אין מצב אשר מבצע סכימה של שני רגיסטר לפקודה שאיננה RTPYPE מבחינת ה OPCODE שלה. RTYPE רגיל לא יכול לגשת לזכרון או ליתר דיוק, המערכת לא תדע לזהות אם זה לא RTYPE במידה וה OPCODE יהיה 000000. לכן היה צורך במצב נוסף. השלמת שאר המצבים היתה פשוטה יותר מבחינת מערכת שליטת המצבים. ניתן היה לקפוץ למצב 3 ומשם להתפצל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

Dispatch ROM 1		
Op	Name	Value
0	R-type	6
2	jmp	9
4	beq	8
35	lw	2
43	sw	2
33	lwr	10

עדכון טבלת ה ROM ממצב 1.

אין עדכון בבקר שינוי המצבים.



State number	Address control action	Value of AddrCtl
0	Increment	3
1	Use dispatch ROM 1	1
2	Use dispatch ROM 2	2
3	Increment	3
4	Go to state 0	0
5	Go to state 0	0
6	Increment	3
7	Go to state 0	0
8	Go to state 0	0
9	Go to state 0	0
10	Increment	3
11	Increment	3
12	Go to state 0	0

ועדכון מערך השליטה.

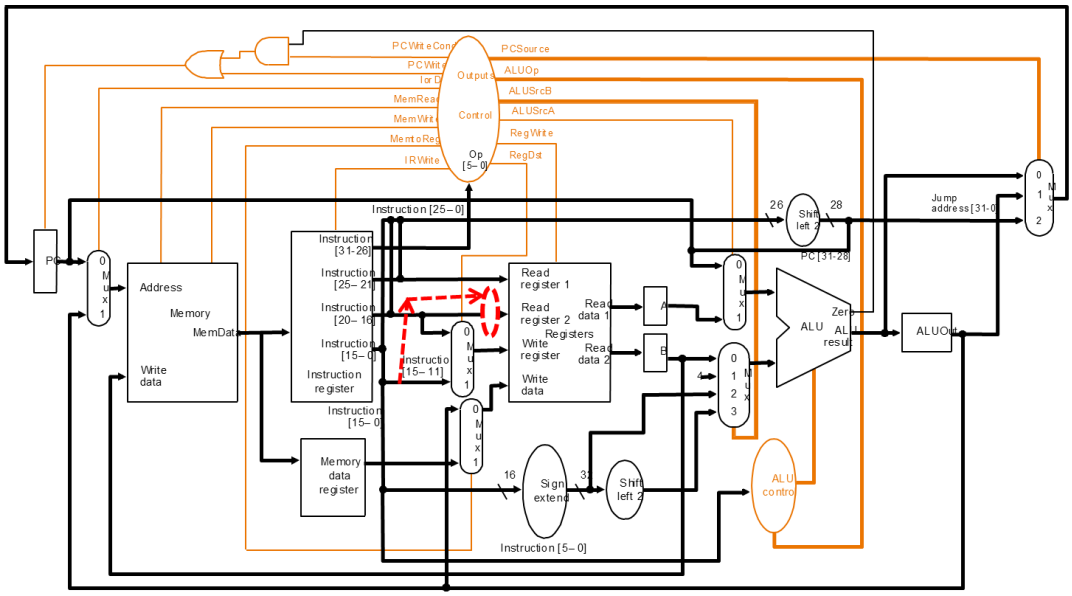
Swr $R1 , $R2, $R3

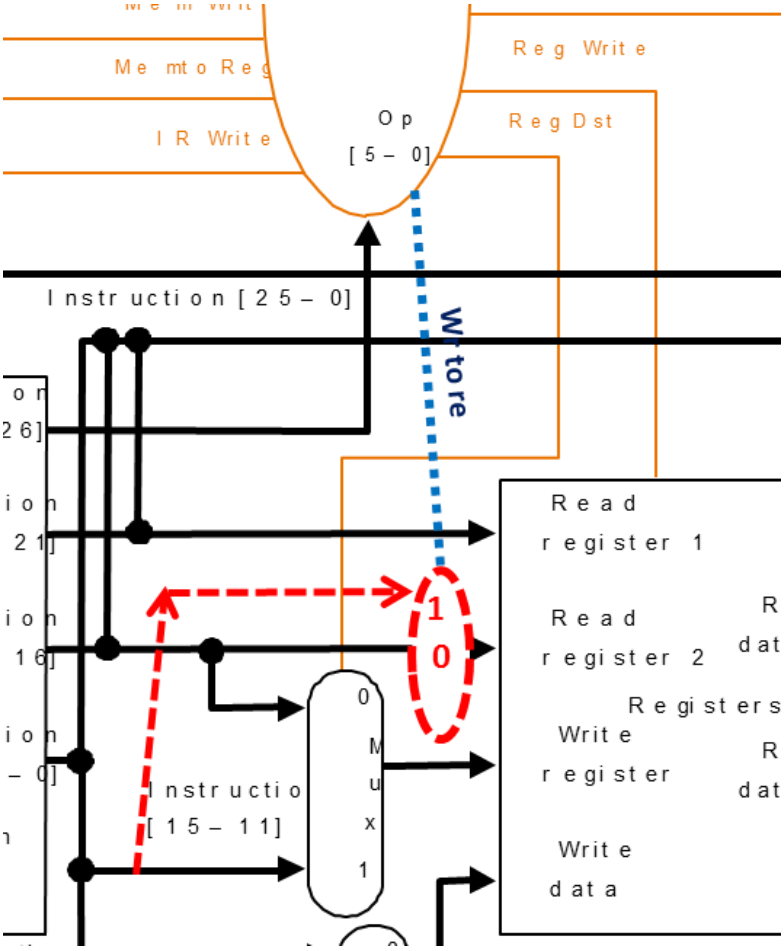
Swr הינה פקודה חדשה אשר מבצעת חיבור בין שני רגיסטרים ומכניסה מילה מתוכן רגיסטר שלישי לזיכרון בכתובת של החיבור. (כיוון הפוך לLWR).

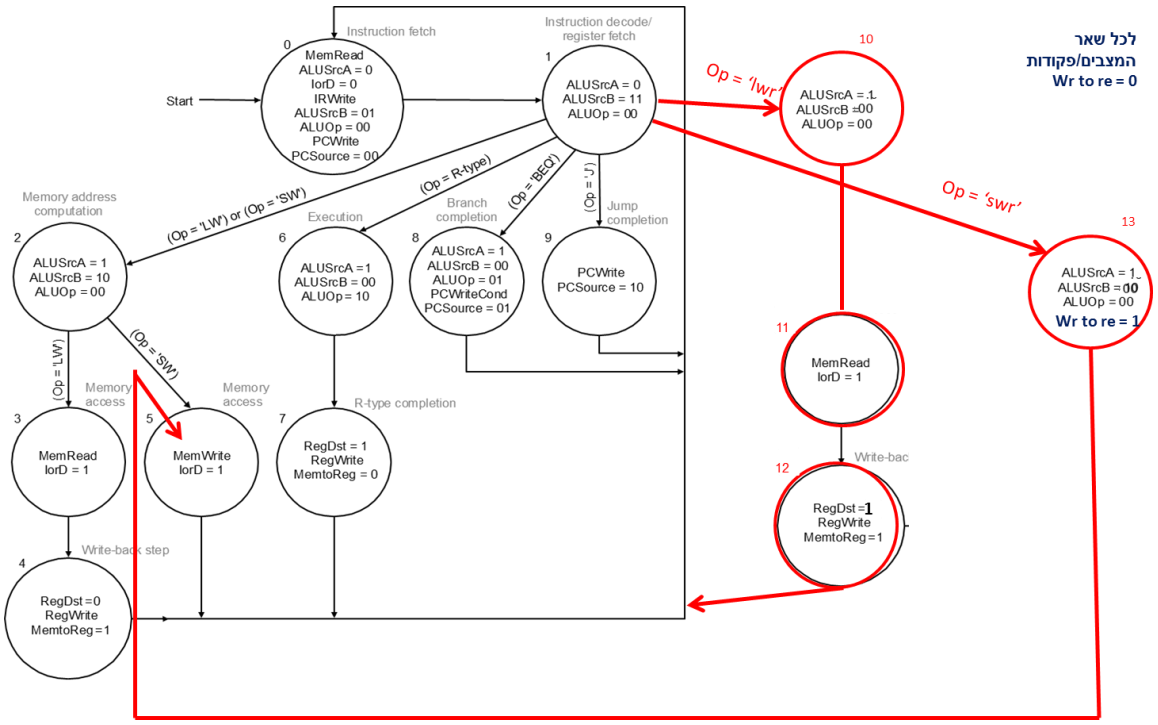
ב. האם הפקודה ניתנת למימוש במערכת single cycle mips ללא שינויים בDATAPATH? הסברו פרטו ונמקו.

ג. 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 ה DATAPATH (כגון הזיכרון או קבוץ הרגיסטרים) על חשבון שינויים אחרים וקוי בקרה נוספים. עדיף להוסיף בוררים מאשר שינויים משמעותיים.

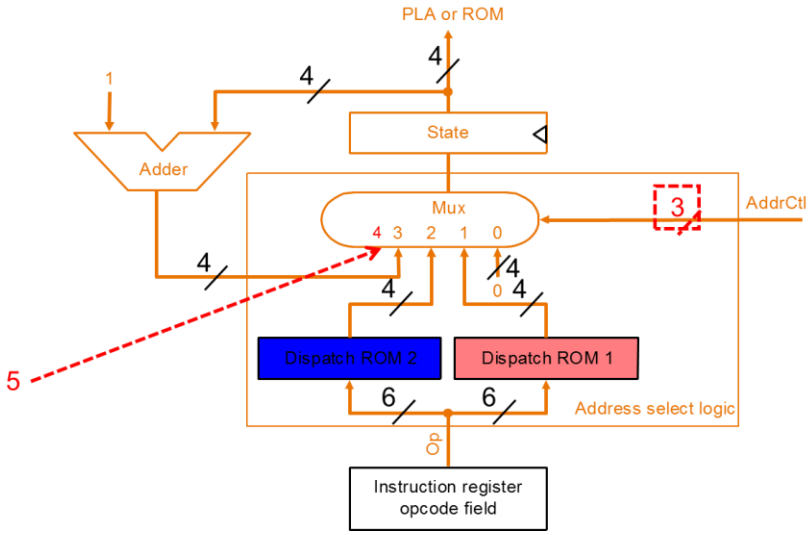
כמעט אותו דבר, צריך להוסיף מצב כדי שיקרא מקובץ הרגיסטרים את הרגיסטר לכתיבה לזכרון בנוסף לחיבור שנעשה באותו זמן בALU. לצורך הקריאה מאופרנד שלישי נצטרך קו בקרה נוסף וmux בכניסה לפורט ה2 של הרגיסטר FILE.




Reg Write
Mem to Reg
IR Write
Op
[5 – 0]
Reg Dst
Instruction [25 – 0]
Write
Read
register 1
Read
register 2
Registers
Write
register
Write
data
1
0
M
u
x
0
1
Instruction
[15 – 11]



Dispatch ROM 1		
Op	**Name**	**Value**
0	R-type	6
2	jmp	9
4	beq	8
35	lw	2
43	sw	2
33	lwr	10
34	swr	13



State number	Address control action	Value of AddrCtl
0	Increment	3
1	Use dispatch ROM 1	1
2	Use dispatch ROM 2	2
3	Increment	3
4	Go to state 0	0
5	Go to state 0	0
6	Increment	3
7	Go to state 0	0
8	Go to state 0	0
9	Go to state 0	0
10	Increment	3
11	Increment	3
12	Go to state 0	0
13	Go to state 5	4

<u>שאלה 2</u> (40 נקודות)

נדרש לחשב סכום של שני ווקטורים $A[0:15]$ ו $B[0:15]$ ולרשום את התוצאה לווקטור $C[0:15]$. $C[i] = A[i] + B[i]$. נתון שהווקטור A נמצא בזיכרון בכתובת 0x40000040 ולאחריו מיד נמצא הווקטור B. את התוצאה יש לרשום בזיכרון בכתובת מיד לאחר הווקטור B. בסיום החישוב יש להפסיק את התכנית ע"י פקודת halt. שימו לב שפקודת halt מתרחשת בשלה ה ID.

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

ראשית, יש להניח שהכתובות $0x40000040$ 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 // $A[0]$ base address
	$t1 = 16 // initialize loop index i
loop:	$t1 = $t1 – 1 // update loop index
	load 0($t0) from mem into $t2 // $A[i] \rightarrow \$t2$
	load 64($t0) from mem into $t3 // $B[i] \rightarrow \$t3$
	$t0 = $t0 + 4 // update base address
	$t4 = $t3 + $t2 // $C[i] = A[i] + B[i]$
	store $t4 in mem at 128($t0) // $\$t4 \rightarrow C[i]$
	bne $t1 $zero loop
	halt

ב. 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.

ג. 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.

ד. 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 // $A[0]$ base address
	$t1 = 16 // initialize loop index i
loop:	$t1 = $t1 – 1 // update loop index
	load 0($t0) from mem into $t2 // $A[i] \rightarrow \$t2$
	load 64($t0) from mem into $t3 // $B[i] \rightarrow \$t3$
	$t0 = $t0 + 4 // update base address
	$t4 = $t3 + $t2 // $C[i] = A[i] + B[i]$
	bne $t1 $zero loop
	store $t4 in mem at 128($t0) // $\$t4 \rightarrow C[i]$
	nop
	halt

ה. 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.

ו. עבור המעבד מסעיף ה' נדרש לחשב את הפעולה $D = 16 * C[15]$. את הערך D יש לרשום בזיכרון בכתובת מיד לאחר הווקטור C. 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 $C[15]$ לעצמו.

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

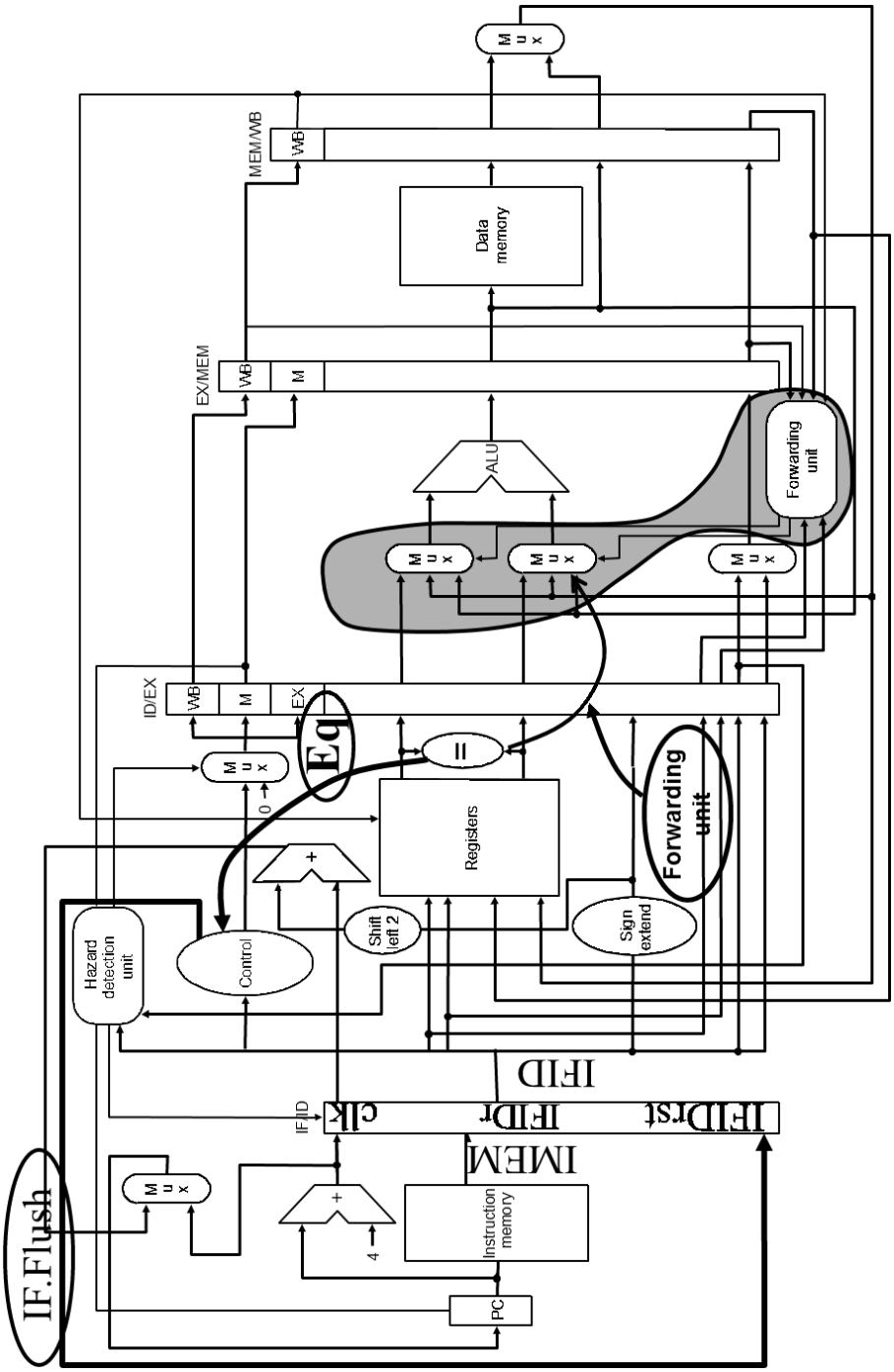
במקרה של שני הפתרונות האחרונים יש לשים לב שמדובר במעבד שבו בוטל IF.Flush, ולכן יש להרחיק את פקודת halt מבדיקת תנאי סיום הלולאה ע"י delay slot, אחרת halt יבוצע מייד בתום הסבוב הראשון.

	load 188($t0) to $t1 // $C[15] \rightarrow \$t1$
	nop
	$t1 = $t1 << 4 // X16 multiply by left shift
	store $t1 in 192($t0) // $\$t1 \rightarrow D$
	nop
	halt

שאלה 3 (30 נקודות)

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

א. רשום תכן בשפת Verilog שמייצר סיגנל זה (בתוך מודול ה MIPS 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
```

[redacted]

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

 …

…
```

[redacted]

```
endmodule;
```