אוניברסיטת בר-אילן

הפקולטה להנדסה

BAR-ILAN UNIVERSITY

Engineering Faculty

ארכיטקטורת מחשבים

תשע"ו סמס' א' מועד א'

83-253

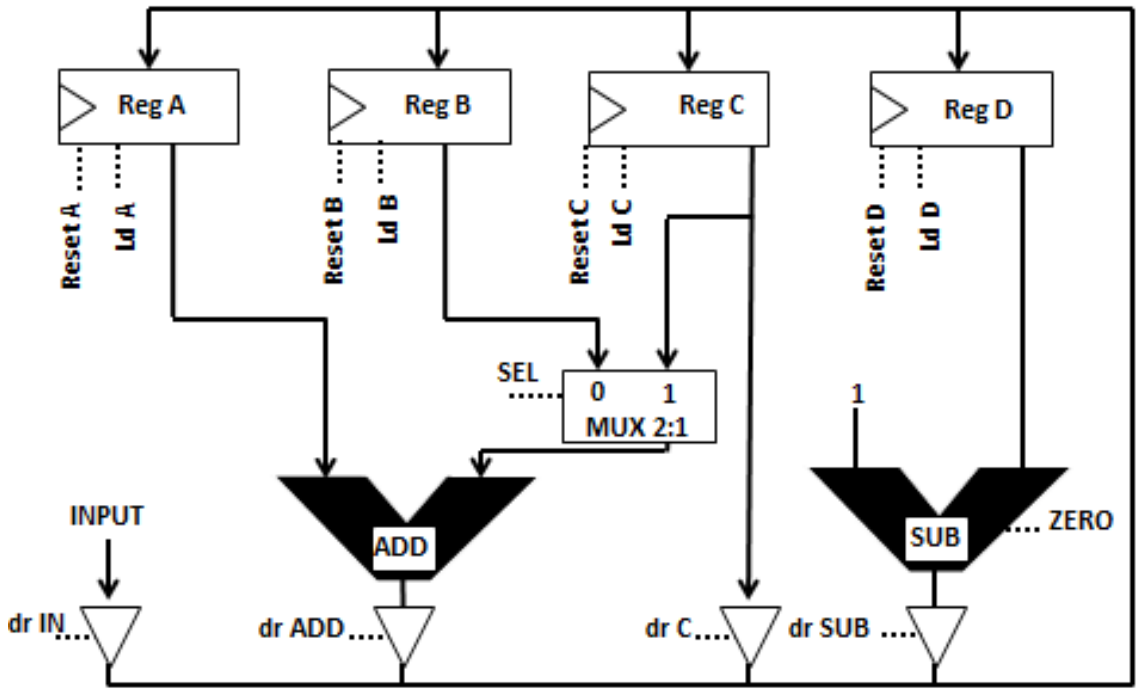
- מרצה : פרופ' שמואל וימר
- מתרגלים : מר יוסף לונדון מר איתמר לוי
- **חומר עזר מותר:** מחברות/ דפים מהרצאות.
- משך המבחן:

  שלוש שעות.
- חובה לענות על כל השאלון.
- **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)

נתונה מערכת סינכרונית כמופיע באיור הבא:



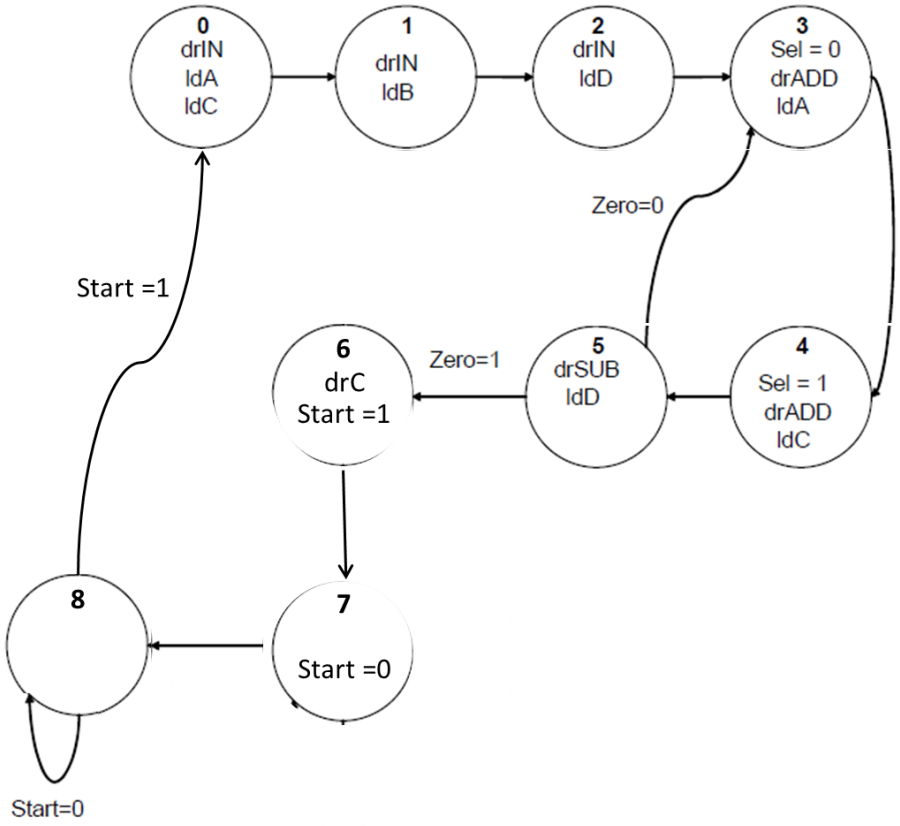
כל הרגיסטרים מחוברים לאותו שעון. קוי הבקרה מסומנים בקוים מקווקוים (למעט קו חיווי הZERO). כמתבקש, הדרייברים הינם דרייברים של TRISTATE.לכל רגיסטר קו בקרה של Ld שנועד לבצע העמסה של מילה במידת הצורך, וקו בקרה של RESET שמאפסת את תוכן הרגיסטר במידת הצורך. קו הבקרה SELECT (SEL) בורר בין 2 מצבי סכימה. קו החיווי ZERO מוציא חיווי של 1 כאשר החיסור ב ALU שממנו יוצא הינו 0. ניתן להניח קו בקרה נוסף START בכדי לומר למערכת להתחיל (כפי שעשינו בתרגול). התוצאה תישמר ברגיסטר C בסיום ויועלה ל BUS התחתון וקו הבקרה START תהווה אינדיקציה לכך שהתוצאה רלוונטית לחישוב שהתבצע.

סדרה חשבונית מוגדרת בצורה הבאה: $a_n = a_{n-1} + d$. בתחילה המערכת תקלוט את האיבר הראשון של הסדרהa_0. לאחר מכן את ההפרש d. ולאחר מכן את N (סוכמים N+1 איברים עד האיבר הa_N).

א. שרטטו דיאגרמת מצבים שתחשב את הסכום של סדרה חשבונית עד למספר איברים N מסוים. הקפידו לרשום בכל מצב אילו קוי בקרה פעילים.

הערות והנחיות: חישבו אילו פרמטרים צריכים להישמר במערכת באופן קבוע ואילו צריכים להשתנות. שימו לב למספר הרגיסטרים הקיימים. נסו לרשום לכל שלב ושלב מה אתם עושים ולמה. שימו לב לכל חלקי המערכת ולמה ניתן להשתמש בהם.

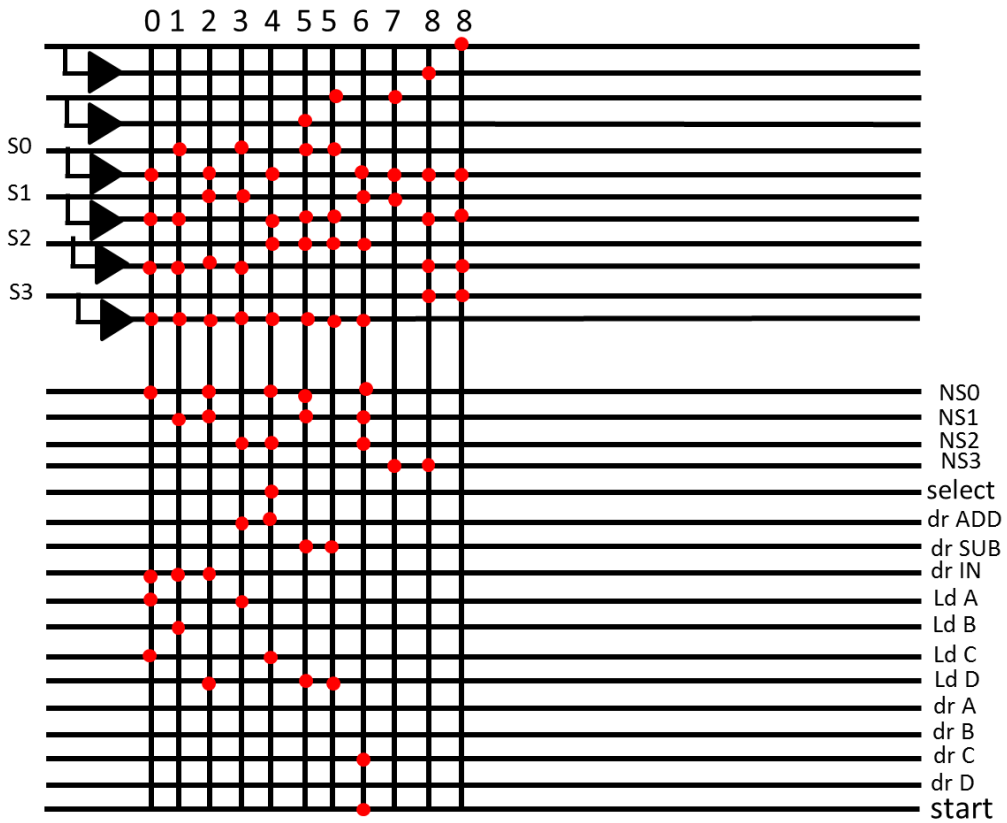
המערכת ממומשת עם התוספת שניתנה במהלך המבחן לגבי קו הבקרה START שיסמן סיום. נתבקשתם להעלות את הקו ל-1 ולאחר מכן להוריד ל-0 שוב ולחכות עד להתחלה חדשה. לא הורדו נקודות על זה כלל.



cycle	REG A	REG B	REG C	REG D
1	a_0		a_0	
2		d		
3				N
4	$a_0 + d$			
5			$a_0 + d + a_0$	
6				$N - 1$
7	$a_0 + d + d$			
8			$a_0 + d + d + a_0 + d + a_0 = a_0 + a_1 + a_2$	
9				$N - 2$

מימוש עבור N=2.

ב. ממשו את הבקר בעזרת PLA או בעזרת טבלת ROM שתשתמש במערכת לעיל לצורך החישוב הנ"ל.



דגשים ל PLA: במישור ה AND היכן שאין נקודה כלל, לא נכנס למכפלה (כלומר בין אם יש לקו 0 או אם יש 1 לא רלוונטי למכפלה (אין שם טרנזיסטור)). במידה ורוצים לדייק יותר ניתן להכפיל את העמודות הללו ולכתוב בין לזה ובין לזה.

ג. כתבו תוכנית קוד באסמבלי שממשת את הסכימה הנ"ל כאשר:

N שמור בזיכרון הנתונים וברגיסטר $t1 שמור הכתובת שלו בזיכרון הנתונים.

a_0 שמור בזיכרון הנתונים וברגיסטר $t2 שמור הכתובת שלו בזיכרון הנתונים.

d שמור בזיכרון הנתונים וברגיסטר $t3 שמור הכתובת שלו בזיכרון הנתונים.

את התוצאה הסופית יש לשמור ברגיסטר $t7$.

הקפידו להגדיר איזה רגיסטרים מבצעים איזה פעולה והסבירו היטב.

(הנחה:$N=0$ הכוונה לאיבר יחיד). היו עוד המון פתרונות. זהו לא הקוד הכי יעיל.

Begin:	lw	$s1,	($t1)	
	lw	$s2,	($t2)	
	lw	$s3,	($t3)	
	add	$s5,	$ZERO,	$s2
	addi	$s4,	$ZERO,	$s1
	add	$s6,	$ZERO,	$s2
Loop:	slt	$t4,	$s4,	$ZERO
	bne	$t4,	$ZERO,	FINISH
	add	$s5,	$s5,	$s3
	add	$s6,	$s6,	$s5
	addi	$s4,	$s4,	(-1)
	j	loop		

FINISH:	add	$t7	$ZERO	$s6

דוגמא נוספת:

Begin:	addi	$s5,	$ZERO,	1	
	lw	$s1,	($t1)		//counter N
	lw	$s2,	($t2)		//sum
	lw	$s3,	($t2)		//an
	lw	$s4,	($t3)		//d
Loop:	add	$s3,	$ s3,	$ s4	$//a_n = a_{n-1} + d$
	add	$s2,	$ s2,	$ s3	$//sum = sum + a_n$
	sub	$s1,	$ s1,	$ s5	$//count = count - 1$
	bne	$s1,	$ ZERO,	Loop	

S1 : N, S1:a0, S3:d, S4:N-I, S5:an, S6:sigma(an)

ד. כמה מחזורים תיקח התוכנית בsingle cycle mips וכמה מחזורים בmulti cycle mips.

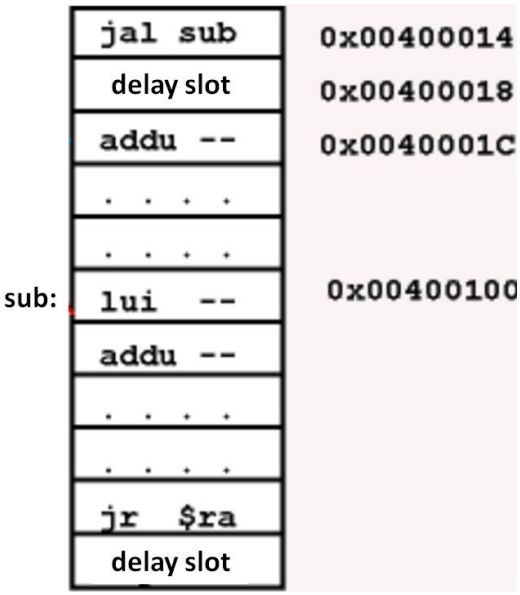
ב single cycle mips ייקח 7 לפקודות הראשונות והאחרון ועוד 6+2(N+1). כלומר $6N + 15$ במולטי, צריך לחלק כל פקודה למשקלו.

שאלה 2 (40 נקודות)

הפקודה jal sub משמשת ב MIPS לקפיצה לשגרה (subroutine) המתחילה בכתובת sub, ומסתיימת בפקודה jr $ra , כאשר באוגר $ra נמצאת כתובת החזרה להמשך הביצוע של התכנית לאחר סיום בצוע השגרה. פקודות Jump כנ"ל מתרחשות בשלב ה ID, ואלו פקודת Jump רגילה איננה קיימת במעבד שלנו.

תזכורת: $ra הינו אוגר מיוחד $ra=31 השמור לצורך אכסון כתובת החזרה בסיום בצוע שגרה.

לפניך קוד Assembly לדוגמא הנוצר ע"י הקומפיילר עבור pipelined MIPS. שימו לב שהקומפיילר מקצה delay slot אחרי הפקודות jal ו jr. ב delay slot תוצב פקודה מועילה אם הקומפיילר מצא כזאת, ו nop אם לא נמצאה פקודה שכזאת.



1. הסבר במדויק מדוע נדרש delay slot?

הדרישה נובעת מכך שכשפקודת jal מתגלה, כבר נכנסה ל pipeline פקודה נוספת. על מנת למנוע את בזבוז מחזור שעון כתוצאה מהשמדתה, הקומפיילר מנסה למצוא איזושהי פקודה לפני ה jal שאפשר להעביר אותה מיד לאחר ה jal. במדה והקומפיילר מצליח, אין הפסד של מחזור שעון. במדה ולא, הקומפיילר מציב ב delay slot פקודת nop.

2. מה צריך להיות ערכו של $ra עבור הדוגמה הספציפית הנ"ל? הסבר מדוע.

<$ra> = 0x00400014 + 0x8 = 0x0040001C

בגלל ה delay slot צריך ה $ra לקבל את address(jal)+8, כלומר להוסיף 4 נוספים לערכו של ה PC שהוא כבר ממילא address(jal)+4. לו הדבר לא היה קורה ו $ra היה מקבל address(jal)+4, אזי הפקודה ברשומה ב delay slot הייתה מתבצעת פעם נוספת. זה לא נורא לו היה שם nop, סה"כ פקודה מיותרת שלא עושה דבר. אולם שימוש ב delay slot לפקודה מועילה עשוי להיות הרסני. הדבר מתרחש באופן אוטומטי, שכן במחזור ה IF של הפקודה הנמצאת ב delay slot מתקיים PC=(address(jal)+4)+4.

3. נדרש מימוש בחומרה של הפקודות jal ו jr המתאים לקוד כנ"ל. צור משאבי חומרה נוספים במדה ונדרש ושנה קיימים במדה ונדרש. שרטטו על גבי שני התרשים המצורפים והסבירו במפורט כל שינוי, תרשים אחד לתוספות ל jal, ושני לתוספות ל jr. שימו לב שפקודת Jump רגילה איננה קיימת במעבד שלנו, ועל כן נדרש גם להוסיף את חומרת הקפיצה.

בפקודה jal יש לדאוג לכתיבת ערך מתאים לתוך $ra ואלו בפקודת jr יש להכניס ערך זה לתוך ה PC. השינויים המפורטים נראים בשרטוט.

4. האם צריך אותות בקרה נוספים, ואם כן, אילו?

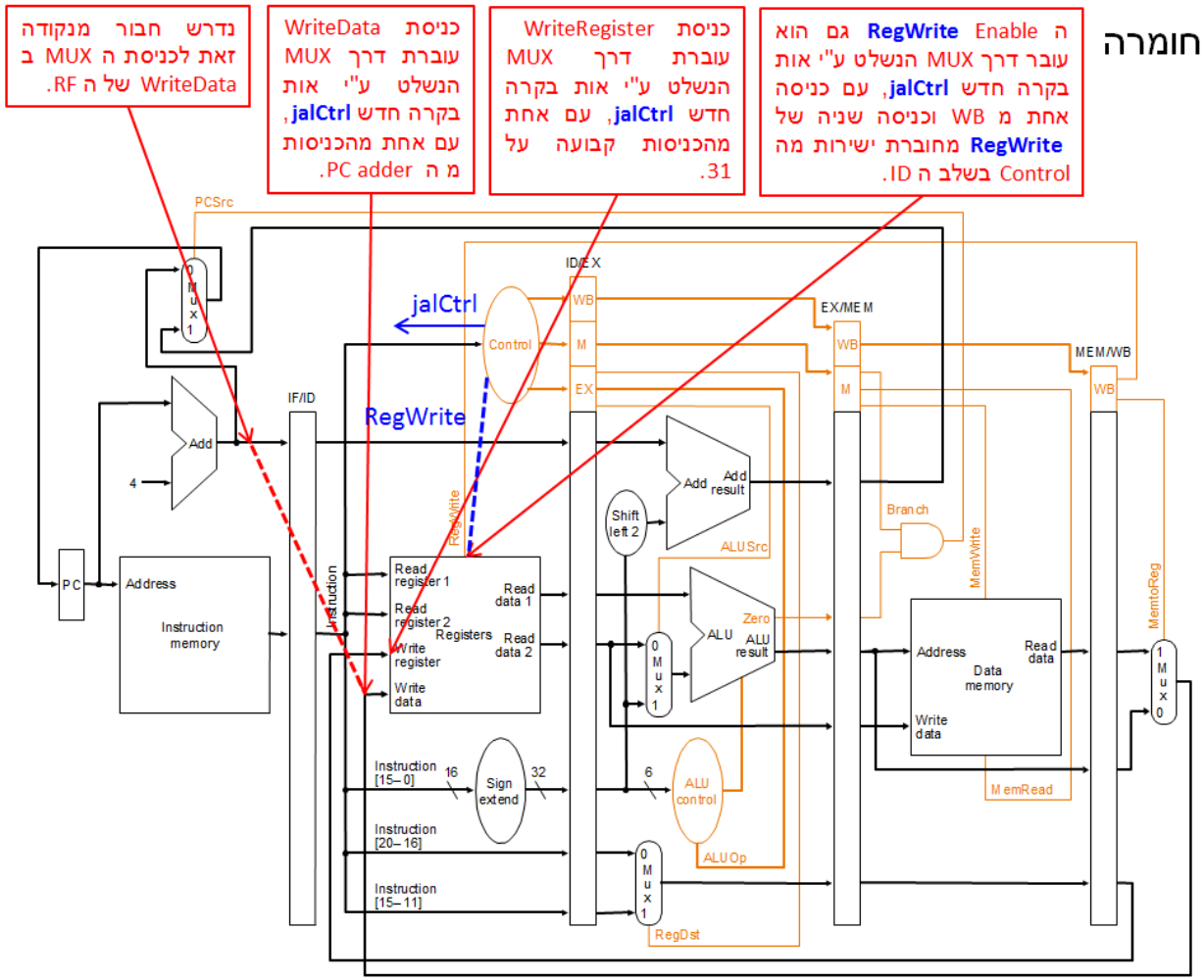
פקודת jal דורשת אות בקרה jalCtrl שתפקידו לגרום לכך ש $ra יכנס ככתובת ל כתיבה לתוך ה RF, וכן ש address(jal)+8 ייכתב ל $ra (ראה תרשים). תפקיד נוסף הוא לגרום לכך שהכתיבה לתוך ה RF אכן תתבצע במחזור השעון הבא.

פקודת jr דורשת אות בקרה jrCtrl שתפקידו לגרום לכך ש $ra יכנס ככתובת לקריאה מתוך RF, ושהערך הנקרא <$ra> ינותב ל PC ככתובת הבאה לביצוע (ראה תרשים).

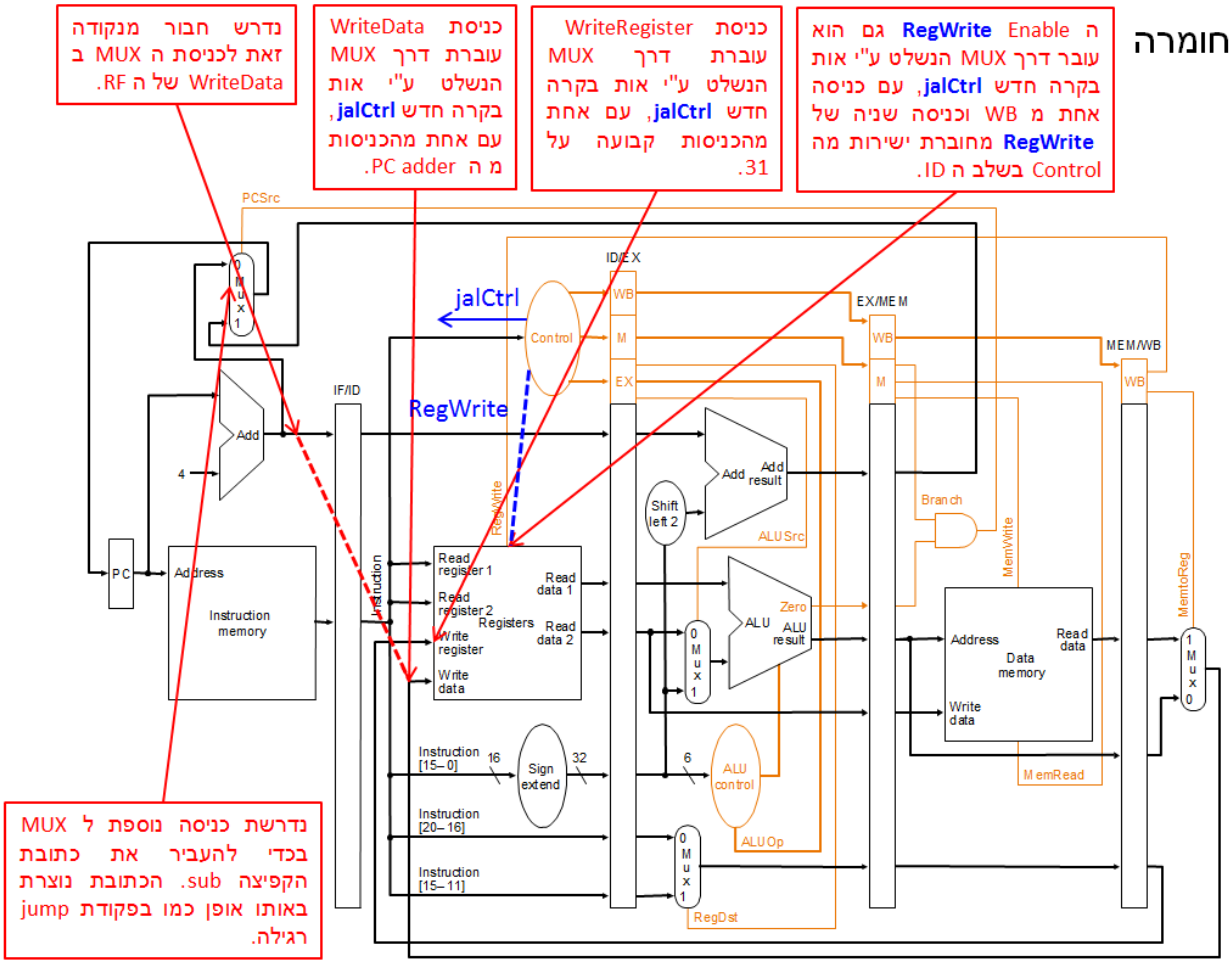
5. האם נדרש עדכון של אותות בקרה קיימים, ואם כן איזה אותות ובאיזה מחזור?
מאחר וכעת ישנה כתיבה ל RF גם בשלב ה ID של פקודת jal, נדרש assertion של ה Enable RegWrite, בנוסף על שלב WB בפקודות הקיימות (ראה תרשים).

6. האם הקוד הנ"ל יעבוד גם ב MULTI CYCLE MIPS? האם נדרש לשנותו? נמקו במדויק את תשובתכם. התייחסו בתשובתכם גם לנכונות וגם ליעילות הביצוע.
ראשית, הקוד יעבוד נכון. כמובן ש delay slot איננו נדרש, משום שמושג זה שייך רק לארכיטקטורת pipeline. במדה וה delay slot משמש לפקודות מועילות, הרי שאין הורדת ביצועים. במדה ויש שם nop, יהיה אבדן זמן של שני מחזורי שעון לכל nop.

תמיכת חומרה ב jal



תמיכת חומרה ב jal



<u>שאלה 3</u> (30 נקודות)

לפניכם מנגנוני קידום הנתונים (forwarding), Fig 1, ו- hazard detection, Fig. 2, שנלמדו בכתה. מנגנונים אלו אמורים לטפל ב- א) data hazard וב- ב) load hazard.

1. רשום תכן בשפת Verilog של כל אחד משני המנגנונים הנ"ל (בסכמאות המתאימות) באיזורים המוקצים לכך באפור למטה. שים לב שאתה מקבל מבנה הצהרות כללי (header) בשפת Verilog של המעבד וסכמה התואמת הצהרות אלו ועליך לשתול פנימה את הקוד הרצוי של כל אחד מהבלוקים בשני הסעיפים. אתם יכולים להשתמש אך ורק בשמות שניתנו על הסכימה באדום (שמוצהרים בקוד). ניקוד ירד על חוסר סדר וחוסר בהסבר\הערות במבנה Verilog \\

להלן קטע קוד assembly של תכנית כלשהי.

...מקרה א:

```
lw $t0, 32($s0)
lw $t1, 36($s0)
bne $t0, $t1, Label_1
...
```

... בהמשך הקוד מקרה ב:

```
add $t0, $s0, $s1
add $t1, $s2, $s3
beq $t0, $t1, Label_2
...
```

2. בהינתן מנגנון הטיפול ב BRANCH (אשר נלמד בשיעור) וניתן ב Fig. 3 האם תכן המנגנונים מהסעיף הראשון מסוגל לטפל נכון בשני המקרים המופיעים בתכנית? נמקו במדוייק את תשובתכם לכל אחד מהמקרים.
3. במידה ונדרש שינוי בתכן ה Verilog, רשמו את השינויים בשפת Verilog באזור המוקצה לכך מטה.

פתרון לסעיף 1:

א) עבור קוד ה forwarding והסכמה המתוארת הצג את תשובתך בחלק זה:

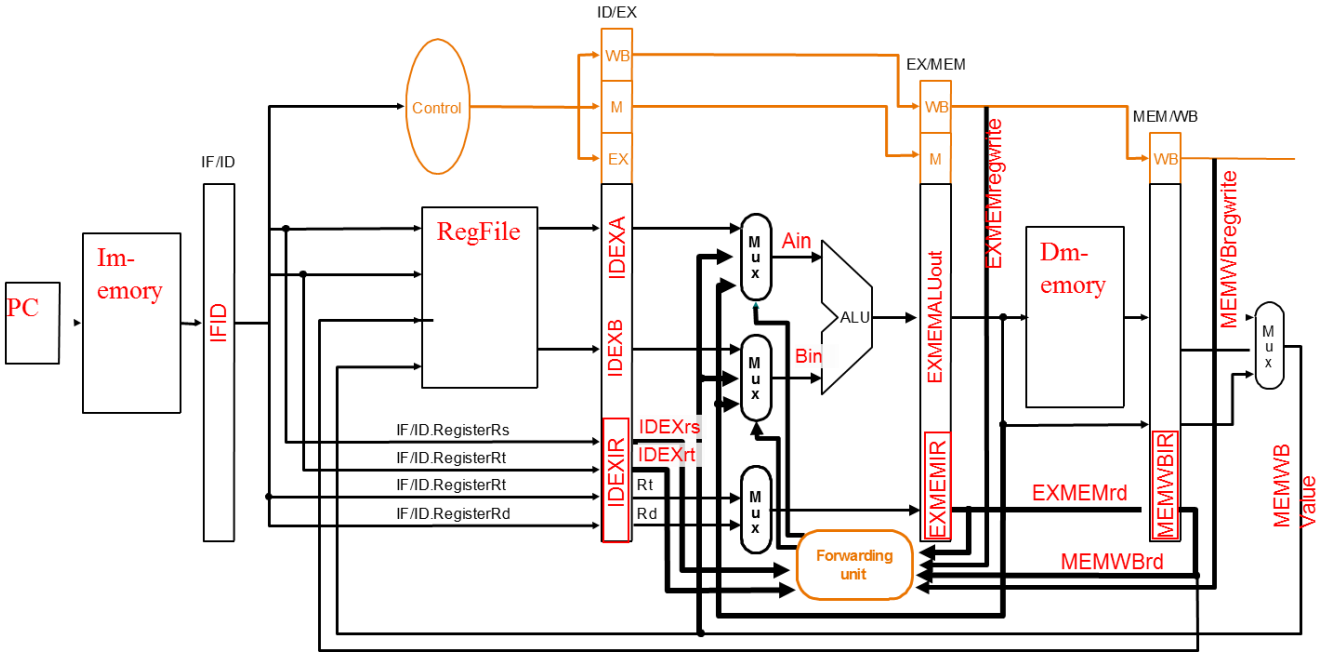


Figure 1

```
module MIPS (input clock);
reg[31:0] PC, RegFile[0:31], IMemory[0:1023], DMemory[0:1023], //memories
    IFID, IDEXA, IDEXB, IDEXIR, EXMEMB, EXMEMALUout, EXMEMIR, MEMWBIR
    // pipeline registers
wire [4:0] IDEXrs, IDEXrt, EXMEMrd, MEMWBrd, MEMWBrt; //wiring register fields
wire  EXMEMregwrite, MEMWBregwrite; //control signals
wire [31:0] Ain, Bin,  MEMWBvalue;
  // ADD DEFINISIONS and DECLERATIONS here. declare the bypass signals
wire bypassAfromMEM, bypassAfromWB,bypassBfromMEM, bypassBfromWB,
bypassAfromLWinWB, bypassBfromLWinWB;
```

```
assign IDEXrs = IDEXIR[25:21]; assign IDEXrt = IDEXIR[15:11]; assign EXMEMrd =
EXMEMIR[15:11];
assign MEMWBrd = MEMWBIR[20:16];  assign MEMWBrt = MEMWBIR[25:20];
```

```
// ADD CODE HERE for the FORWARDING UNIT. Bypass to input A from the MEM stage

assign bypassAfromMEM = (IDEXrs == EXMEMrd) & (IDEXrs!=0) & (EXMEMregwrite==1);

// Bypass to input B from the MEM stage

assign bypassBfromMEM = (IDEXrt == EXMEMrd)&(IDEXrt!=0) & (EXMEMregwrite==1);

// The bypass to input A from the WB stage

assign bypassAfromWB =( IDEXrs == MEMWBrd) & (IDEXrs!=0) & (MEMWBregwrite==1);

// The bypass to input B from the WB stage

assign bypassBfromWB = (IDEXrt == MEMWBrd) & (IDEXrt!=0) & (MEMWBregwrite==1);

// The bypass to input A from the WB stage

assign bypassAfromLWinWB =( IDEXrs == MEMWBIR[20:16]) & (IDEXrs!=0) & (MEMWBregwrite
==1);

// The bypass to input B from the WB stage for an LW operation → this will work only it the alu
operation is more than 2 cycles after the load and the loaded data was not required in these 2
cycles …(lw) otherwise stall\bubble is required

assign bypassBfromLWinWB = (IDEXrt == MEMWBIR[20:16]) & (IDEXrt!=0) & (MEMWBregwrite
==1);

// The A input to the ALU is bypassed from MEM if there is a bypass there, Otherwise from WB if

// there is a bypass there, and otherwise comes from the IDEX registers

assign Ain = bypassAfromMEM? EXMEMALUOut :

(bypassAfromWB | bypassAfromLWinWB)? MEMWBValue : IDEXA;

// The B input to the ALU is bypassed from MEM if there is a bypass there,

// Otherwise from WB if there is a bypass there, and otherwise comes from the IDEX register

assign Bin = bypassBfromMEM? EXMEMALUOut :

(bypassBfromWB | bypassBfromLWinWB)? MEMWBValue: IDEXB;
```

//Assume the REST OF THE MIPS CODE is here:….. you do not need to code it ☺

ב) עבור בלוק ה HAZARD DETECT מספיק לייצר את סיגנל ה STALL ו- IFIDclken, PCclken. רשום הקוד אחר הסכמה.

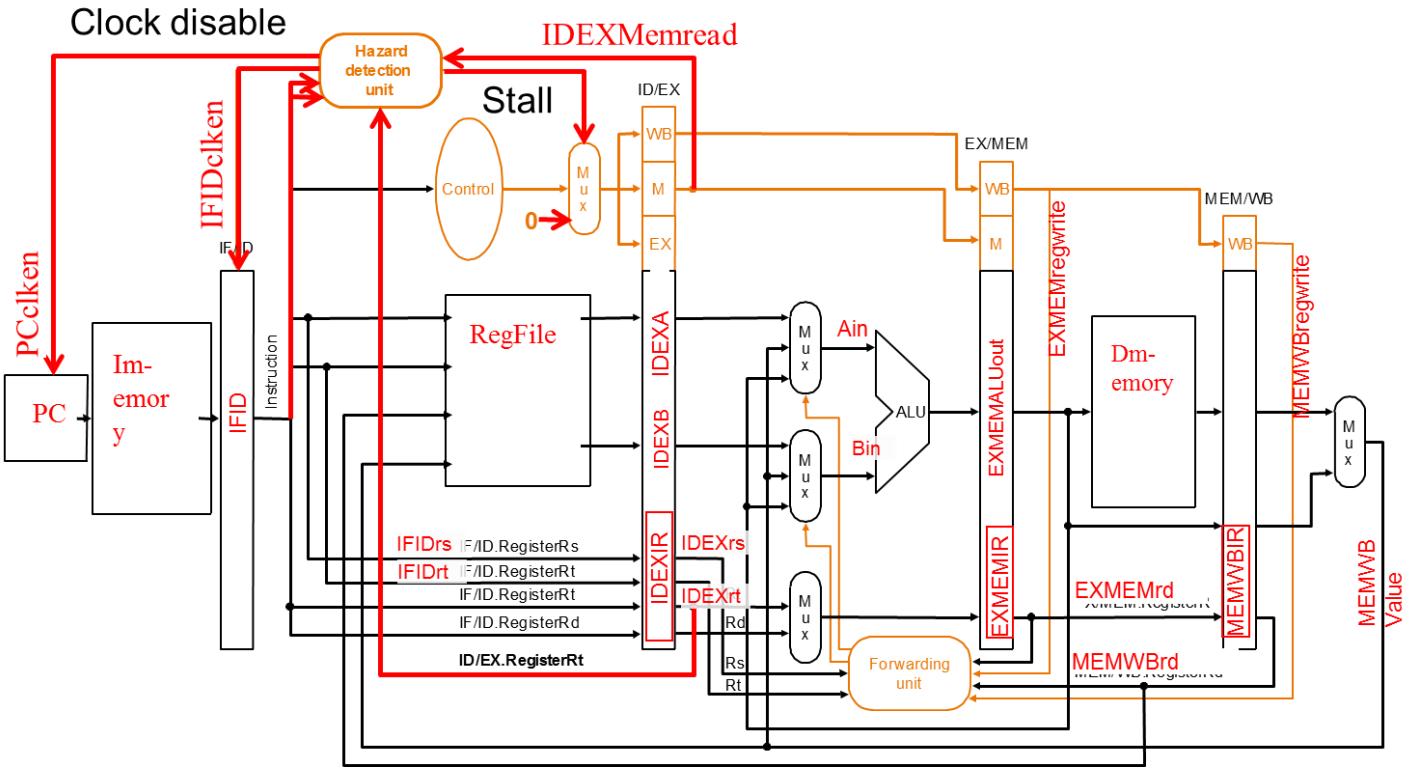


Figure 2

//use the signals from the previous declarations and add additional declarations here from the attached scheme.

// Hazard detection Unit in the ID stage that inserts a stall between the load and its use. ID Hazard Detection:

```
wire stall, IDEXMemread, IFIDclken, PCclken;

wire [4:0] IFIDrs, IFIDrt; \\can assume were previously assigned




assign stall=   (IDEXMemRead &( IDEXrt == IFIDrs | IDEXrt == IFIDrt );

assign PCclken=~stall;

assign IFIDclken=~stall;
```

פתרון עבור סעיף 2:

תשובה: עבור התכן הנתון בFig. 3 המנגנון של forwarding כן מספק את סיגנלי הבקרה לכך שיש לבצע קידום נתונים מדרגת ה EXMEM או MEMWB אך הוא לא בודק אל מול רגיסטרי ההשוואה של BRANCH בשלב ה DECODE ולא מחליף את רגיסטרי ההשוואה ברגיסטרים המתאימים משלבים מתקדמים יותר ב PIPE. גם במקרה א וגם במקרה ב הבעיה קיימת.

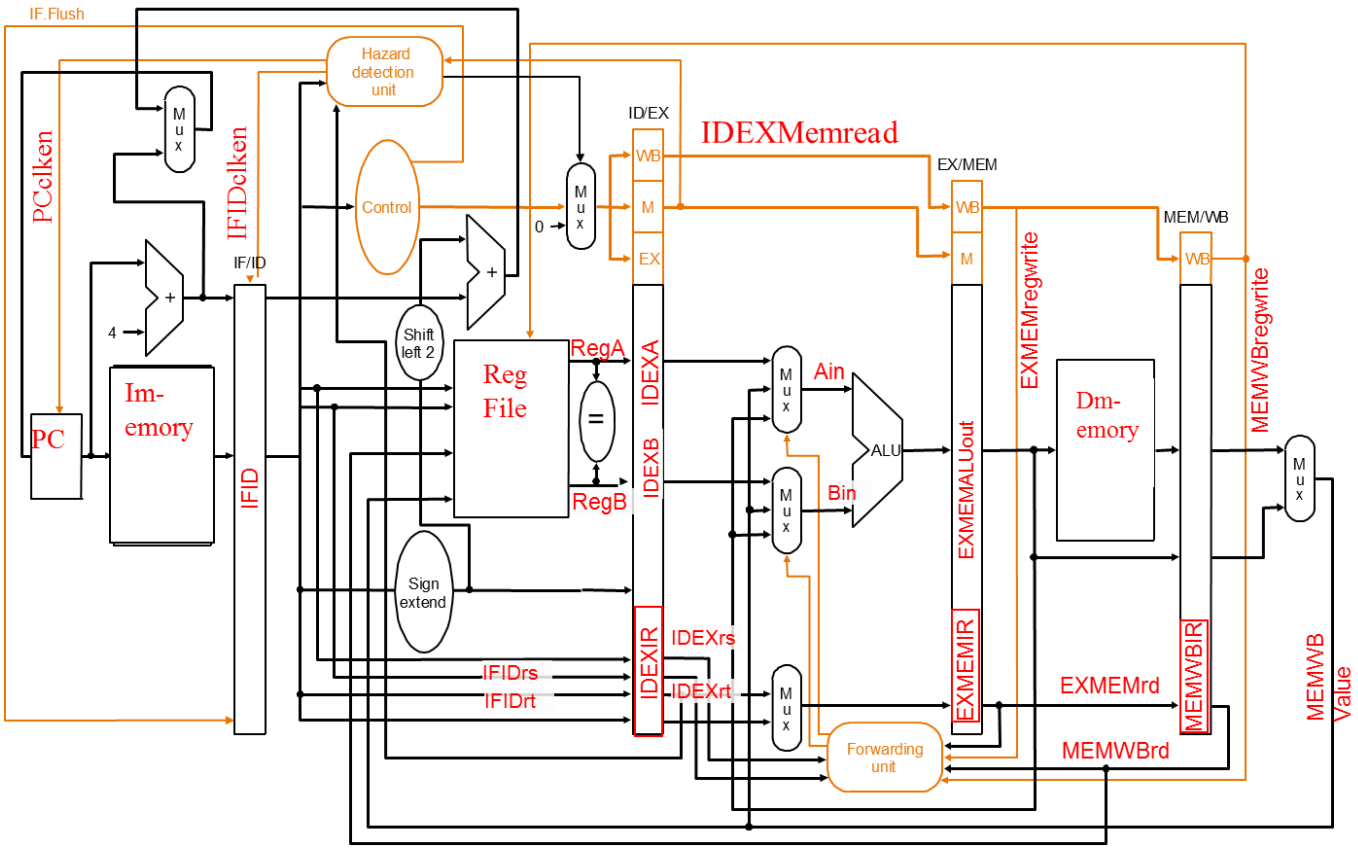


Figure 3

פתרון עבור סעיף 3:

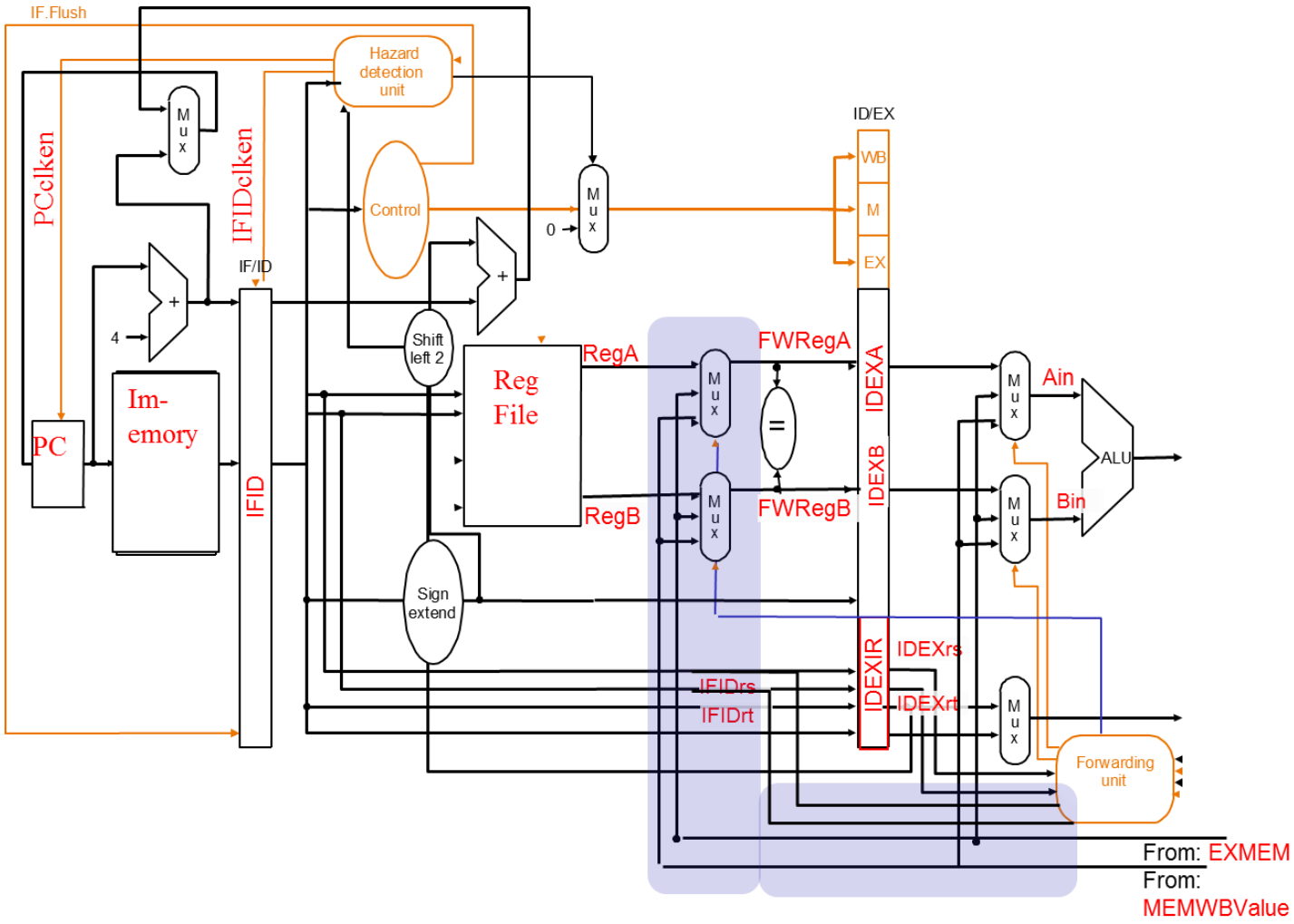


Figure4

הפתרון מציע שמבנה יחידת ה forwarding כפי שנבנה מתאים לטיפול בבעיה. יש לשכפלו בדיוק פרט לשינוי שמות הסיגנלים (בגלל שזו חומרה נוספת או בנייה כמודול ואינסטנסיאציה) והשינויים הבאים בארגומנטים (כפי שמתואר בשרטוט בכחול). הסיגנלים שיש להחליף בקוד הם : , IFIDrd← →IDEXrt IFIDrs← →IDEXr.

מקום לקוד: