

LAB 5

Advanced CPU architecture and hardware
accelerators lab

Ido Ron: 322384330

Binyamin Oumanski: 322688946

הקדמה :

מטרה:

בארכיטקטורת single cycle.pipeline מ עברו ל pipeline מipsם. Throughput קלומר כפולות הייעילות הוא בו הינוBH. יתר גובה יתירה. שיתרנו במאגרים תיאורתיים של מודנו. שילבנו וישמן אלמנטים מעבדה. מחשבים מחשביים. ארכיטקטורת הקורס בהרצאות של מושגים. אסמבלי, שאותן ביצענו בעבודה זו. בסיסית של פקודות אסמבלי, שאותן סימלצנו, בדקנו והעברנו ורפייקציה במודלים. models. FPGA. ואחר מכן מימושנו אותן בחומרה ב

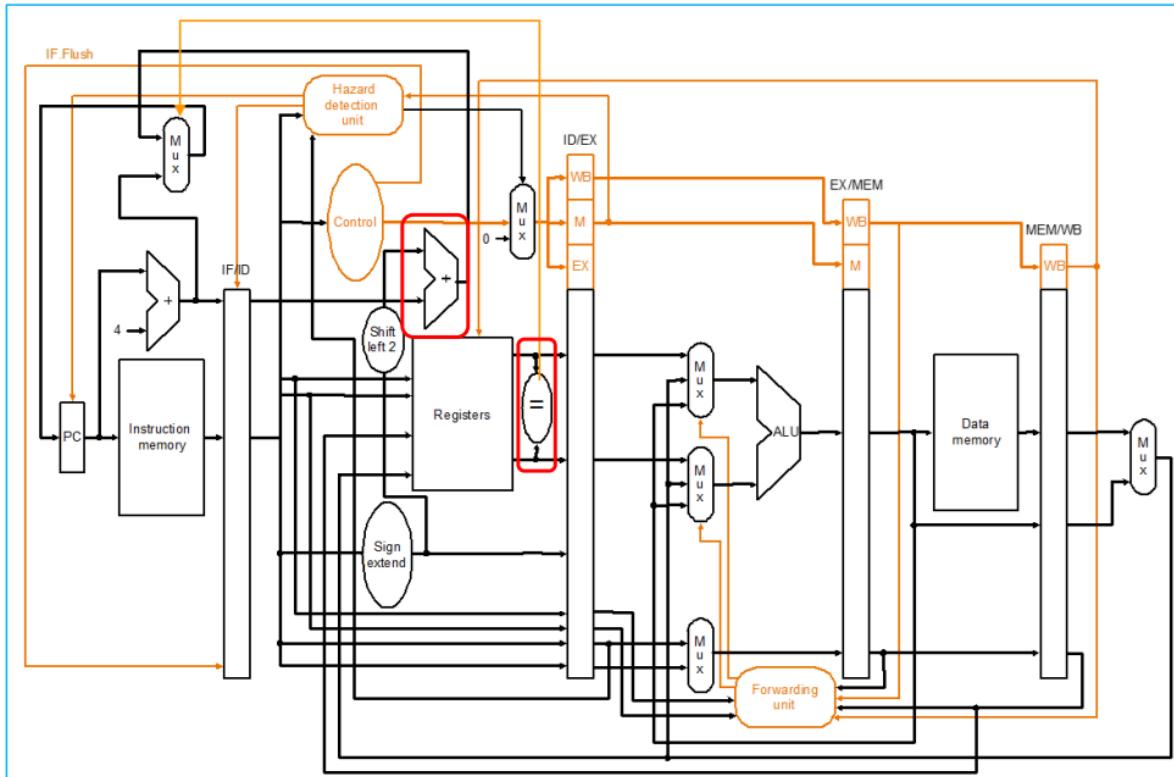


Figure 6: Five-stage pipelined MIPS architecture with forwarding and single delay slot support

מציאת תדר מקסימלי:

נדרשנו למצוא תדר מקסימלי של המערכת , הוא חושב בQUARTUS על ידי מציאת המסלול הרכיבי הארוך ביותר בין שני שעוניים .

התדר המקסימלי שיצא הוא 50.37 MHZ.

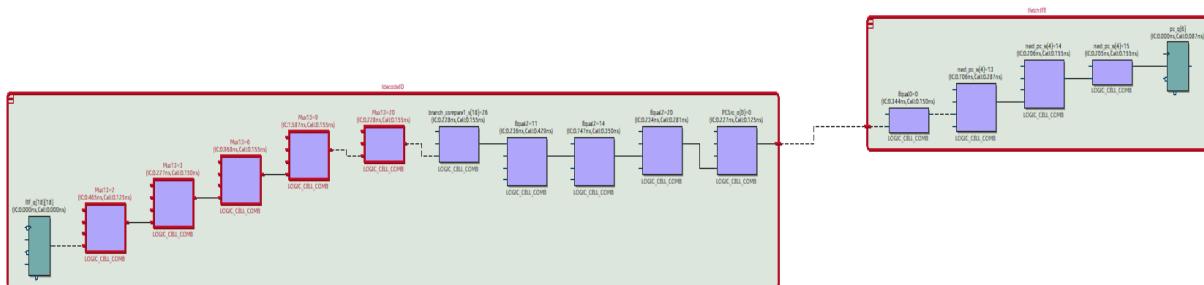
מסלול קרייטי:

	Fmax	Restricted Fmax	Clock Name	Note
1	42.5 MHz	42.5 MHz	altera_reserved_tck	
2	50.37 MHz	50.37 MHz	clk_i	

נתיב קרייטי:

הנתיב הקרייטי מגדר את הזמן הארוך ביותר שדרוש לאות לעبور בין שני איזוריים במחזור שעון אחד ולכן קובע את תדירות השעון המקסימלית. אם הנתיב הקרייטי ארוך מדי, ישתכו הפרות של תנאי setup/hold והתקוד ישתבש. חשוב לשמר על נתיב קרייטי קצר על מנת ליצור מערכת אמינה, יעילה ומהירה.

אצלנו הנתיב הקרייטי הוא בין הDECODE לבין הFETCH.



פירוט של המערכת :

נראה ונסביר על המערכת והמודולים שמרכיבים אותה.

המערכת:

למערכת הוספנו סיג널ים נוספים :

- CLKCNT – כמות עליות השעון

- FHCNT – כמות flushים

- STCNT – כמות stalls

וSIGNEL שמראה מתי עושים בריך פוינט:

BreakPoint – כתובת הנקודות BPADD

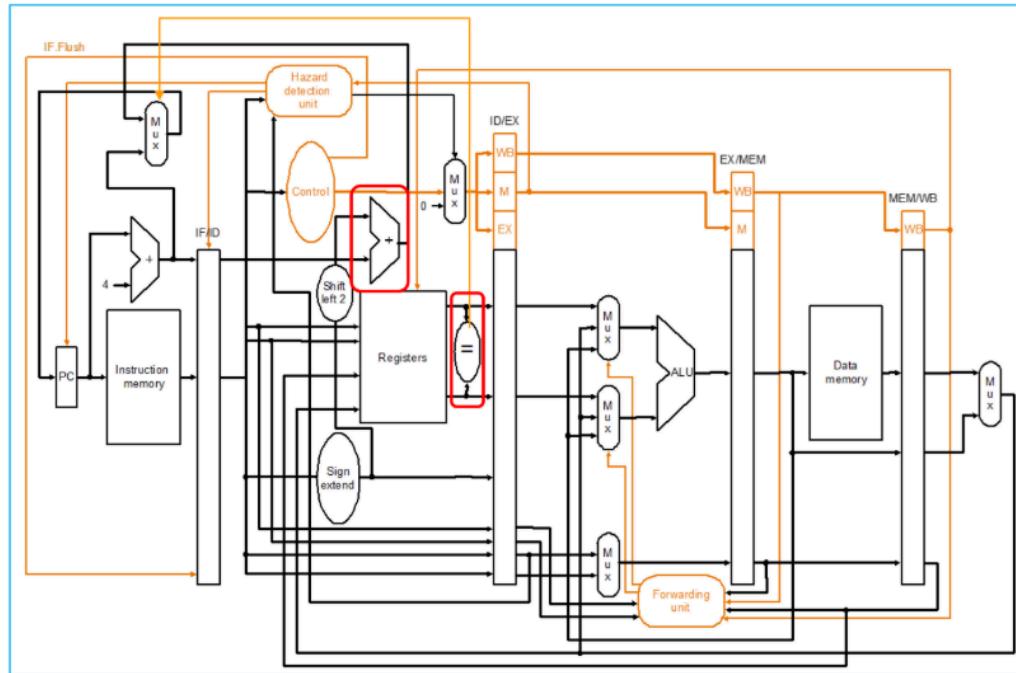
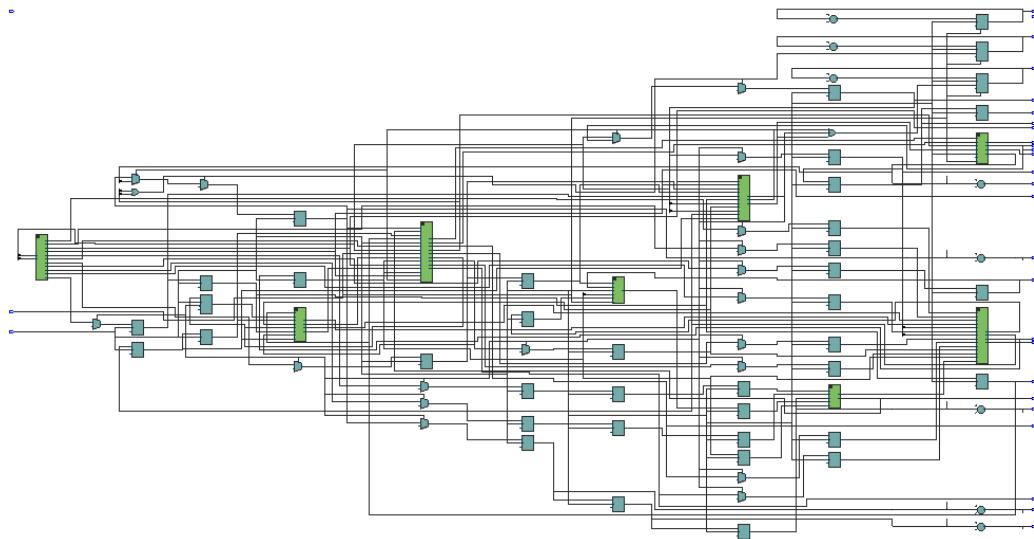


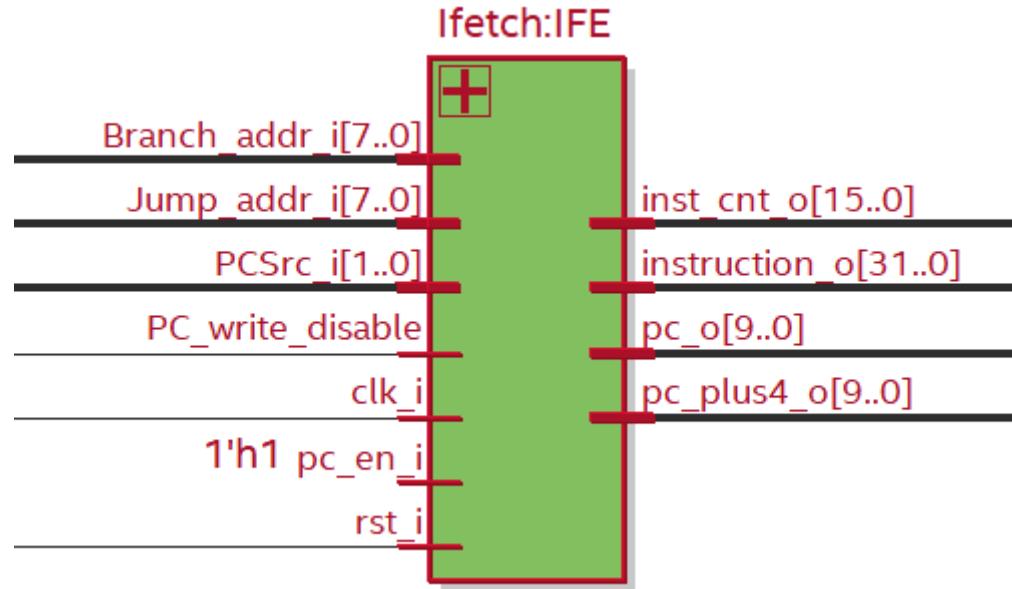
Figure 6: Five-stage pipelined MIPS architecture with forwarding and single delay slot support

תרשים RTL:

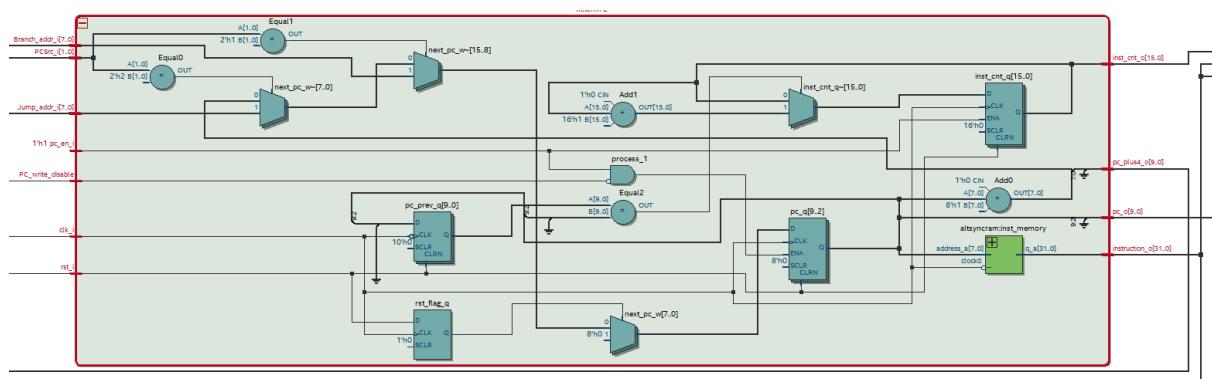


שלב ה-IF

סקירת פועלות המודול
במודול זהה מוצאים את הפקודה בכתובת PC מה-
ITCM.
במימוש ה ישמן בורר שיבחר את כתובת ה-PC
 בהתאם לסוג הפקודה.
האקס בורר בין אופציה של פקודה רגילה, jump ו
.branch



RTL:



שלב ה-ID

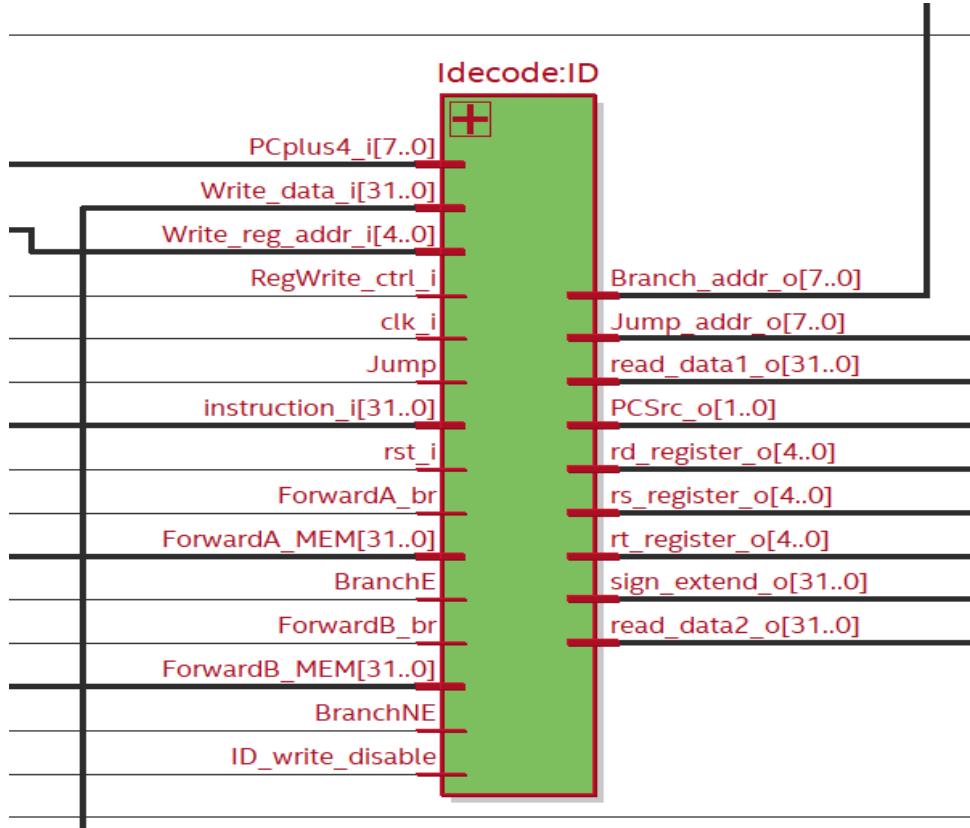
הפקודה עוברת מה IF ל-ID , שם המעבד מסוויג את סוג הפעולה לפי TYPE Jump , סיווג זה מתבצע לפי מבנה הפקודה.

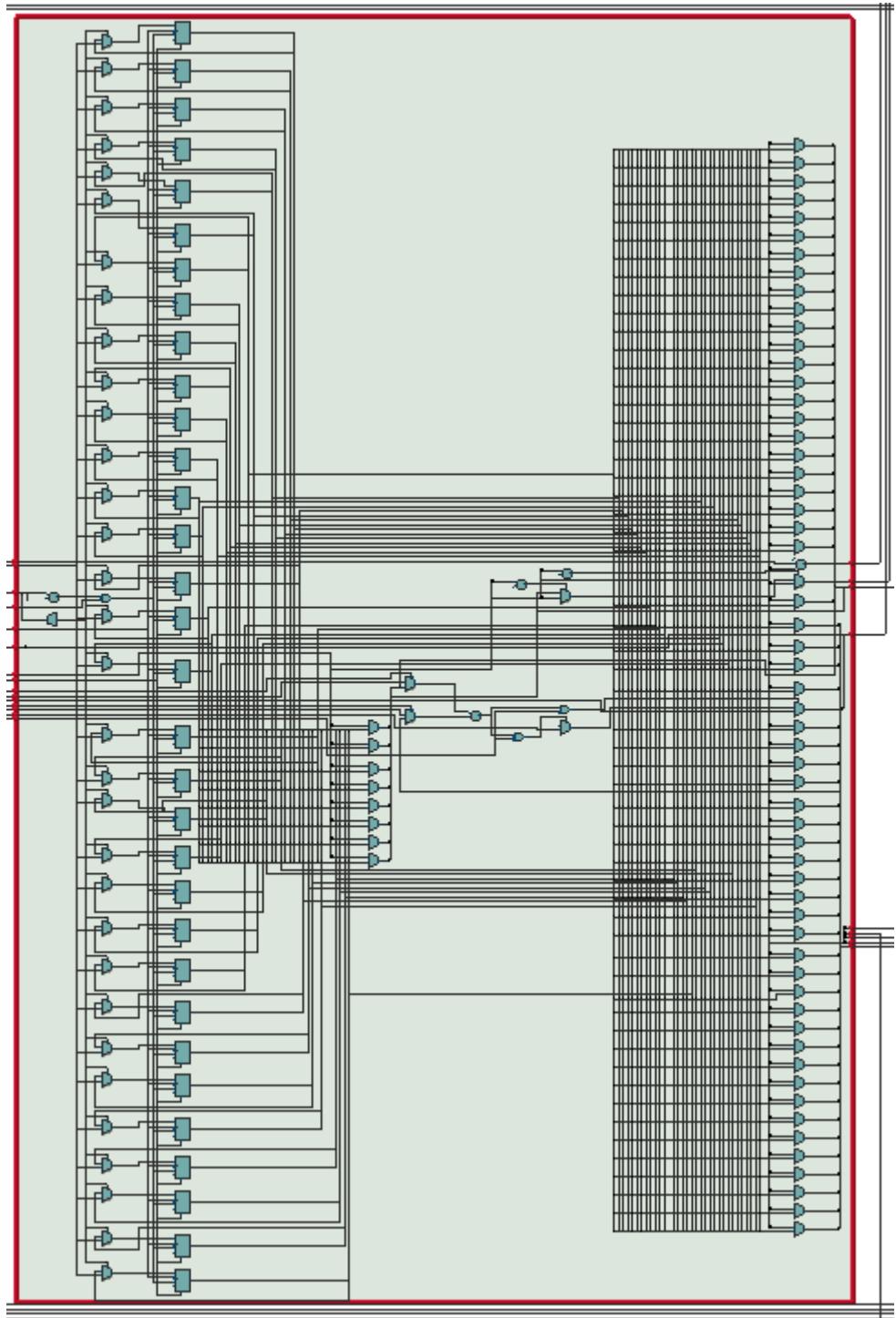
Type	format (bits)						-0-
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)	
I	opcode (6)	rs (5)	rt (5)	immediate (16)			
J	opcode (6)	address (26)					

שלב זה קורה במקביל ל- Control.

במוצא ה RF יש משווה שבעזרתו ניתן לדעת האם ציר לבצע brunch או jump בהתאם לפקודה של המשתמש. במודולים מרכיבת קווי בקרה שבהתאם לסוג הפעולה היא יודעת איזה קווי בקרה להפעיל ולהעביר הלאה לשאר המודולים ב Pipeline .

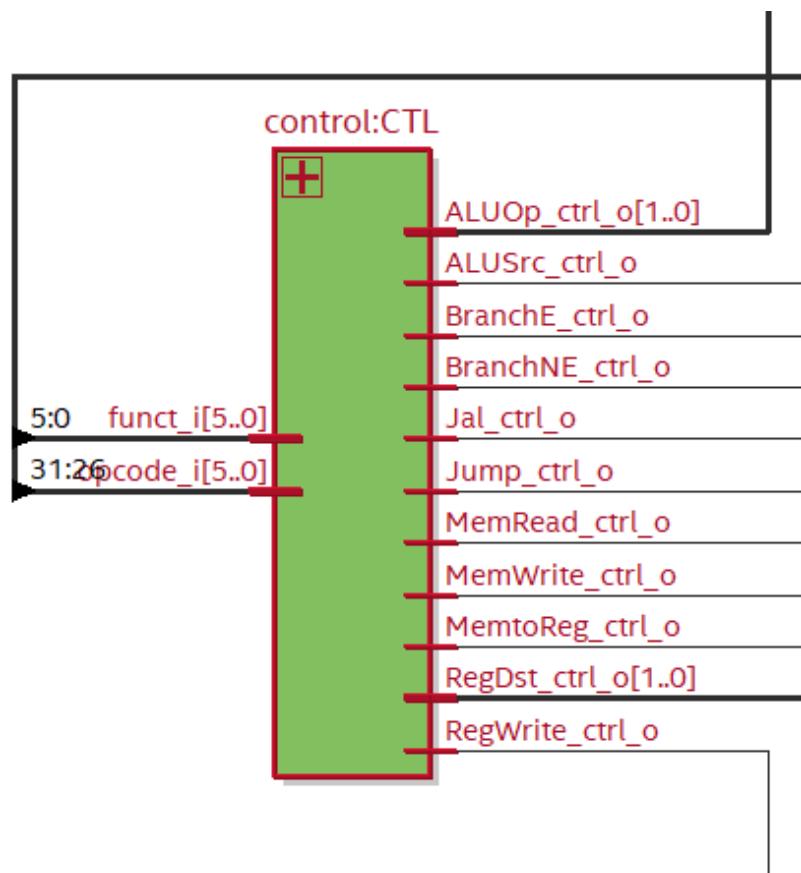
לדוגמא כאשר יש פעולה של כתיבה לרגיסטר , המערכת מדליקת את קו הקריאה של regwrite או לדוגמא כאשר יש פקודה ששיכת לTYPEIT , ידלק קו הקריאה של pcsrc שאחראי במודול הבא לברור את ה immediate .

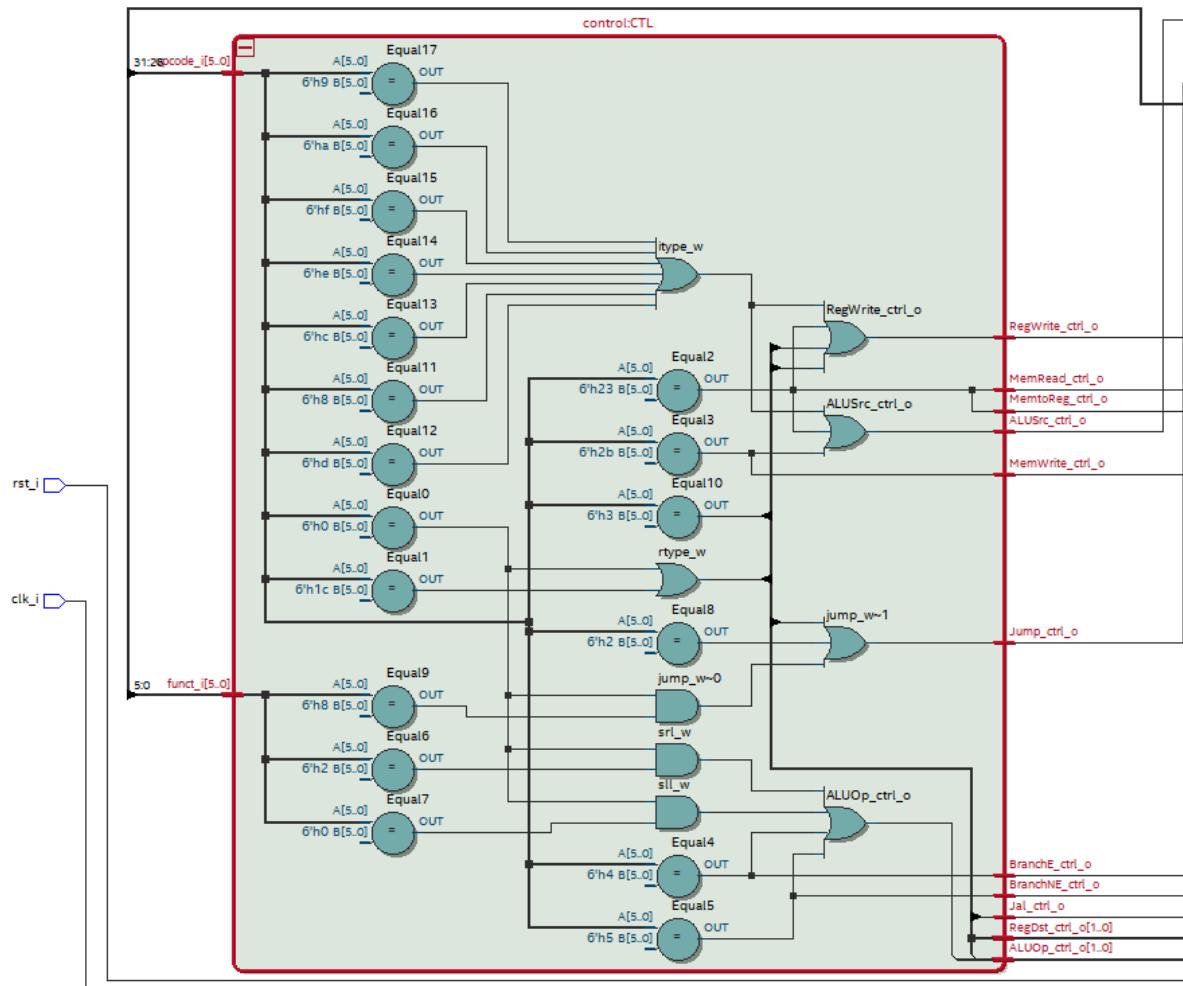




Control

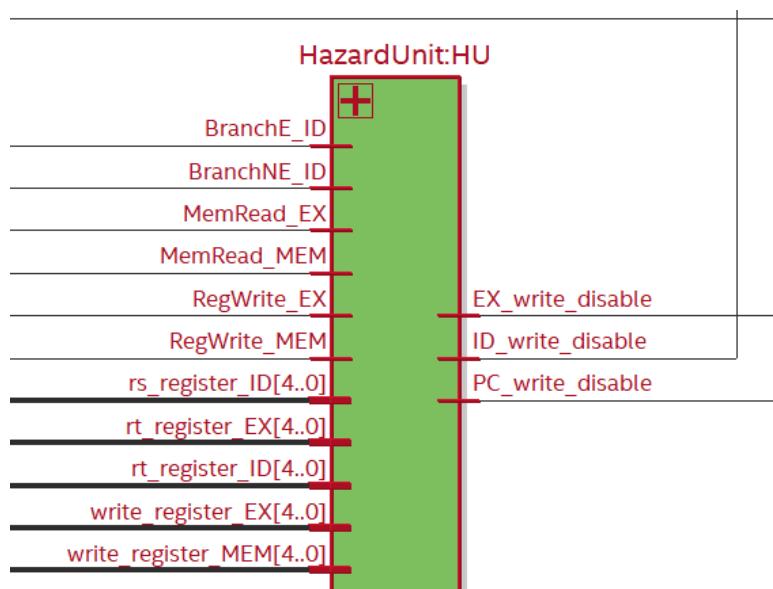
רכיב זה אחראי להזיהוי קוווי בקרה שגם הם רצים ב-Pipeline. קווי הבקירה נדלקים כאשר פעולה מגיעה לשלב ה-ID בהתאם לפעולה. קווי בקרה לדוגמא: כתיבה ל-RF, סלקטורים של XUM, קריאה או כתיבה לזיכרון וכו'.

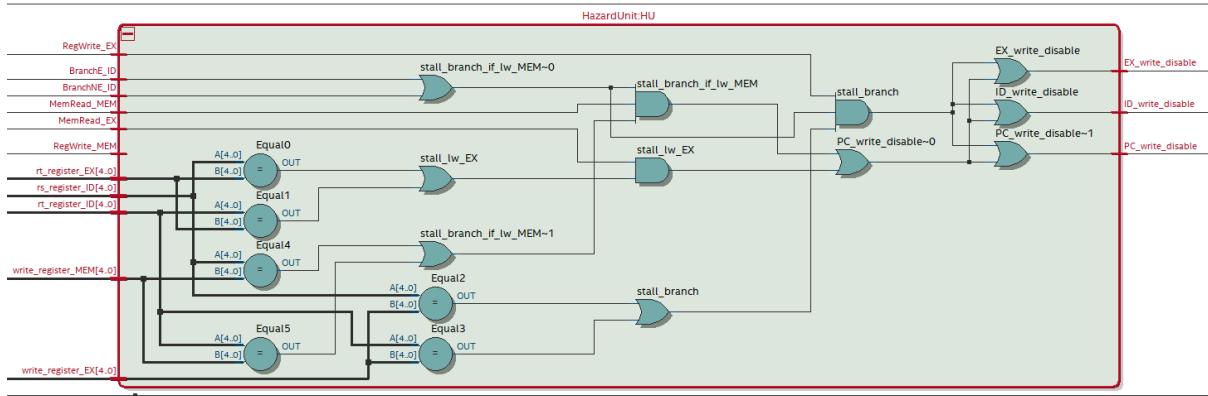




Hazard:

מטרת הרכיב הינה לזהות מצבים המיידם דרשו אך לא מוכן וגם לא ניתן לפתורן ע"י Forwarding. במצבים כאלה הרכיב מוציא קוויב בקרה אשר תפקידם לבצע Stall או IF/ID/EXE Stall גורם ל-PC לעזר, עוצר את רגיסטר ה-ID מלhattעכן מה שגורם לפעולה ב-ID להישאר, ושם פועל NOP בשלב ה-EXE. זה בעצם נותן עוד מחזור לפעולה מסויימת להתקדם הלאה ב-Pipeline בכך שהميدע יהיה מוכן. תפקיד ה-ID הוא "לנקות" את שלב ה-ID במצבים של Branch taken או Jump.

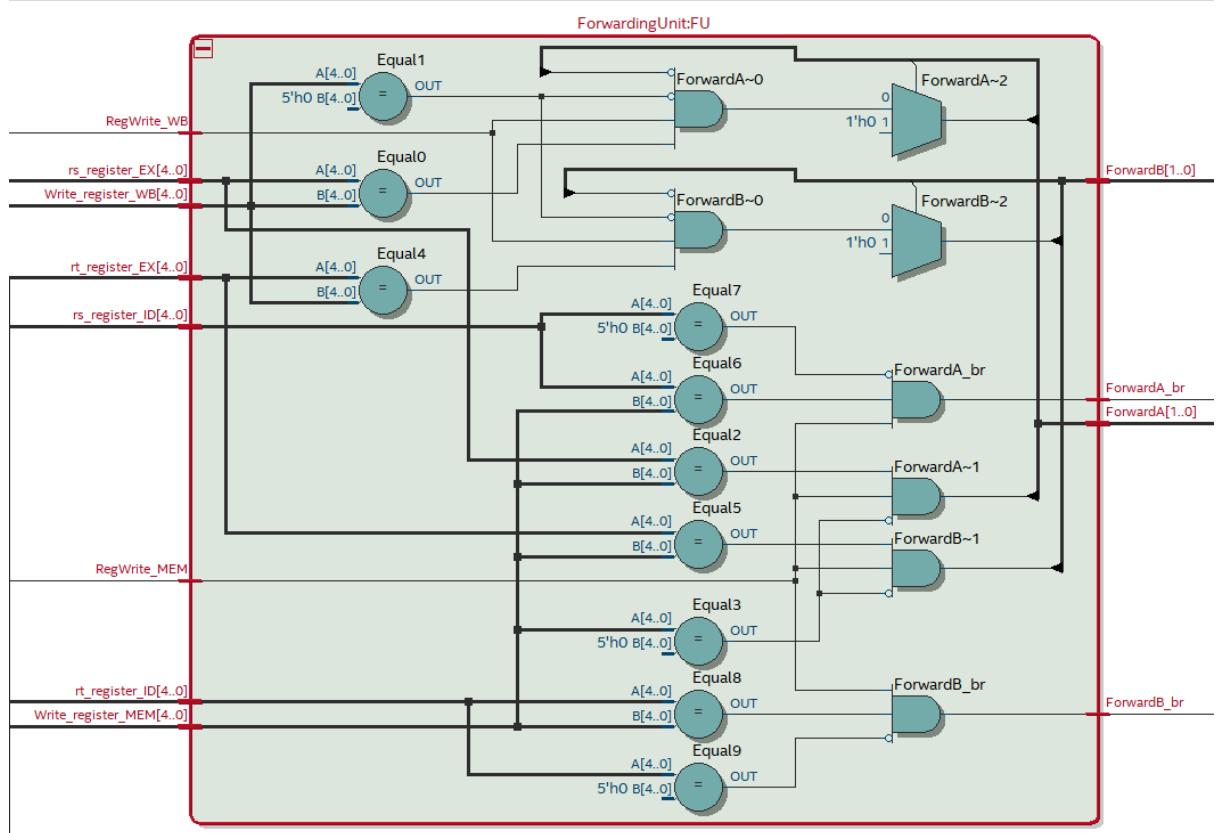
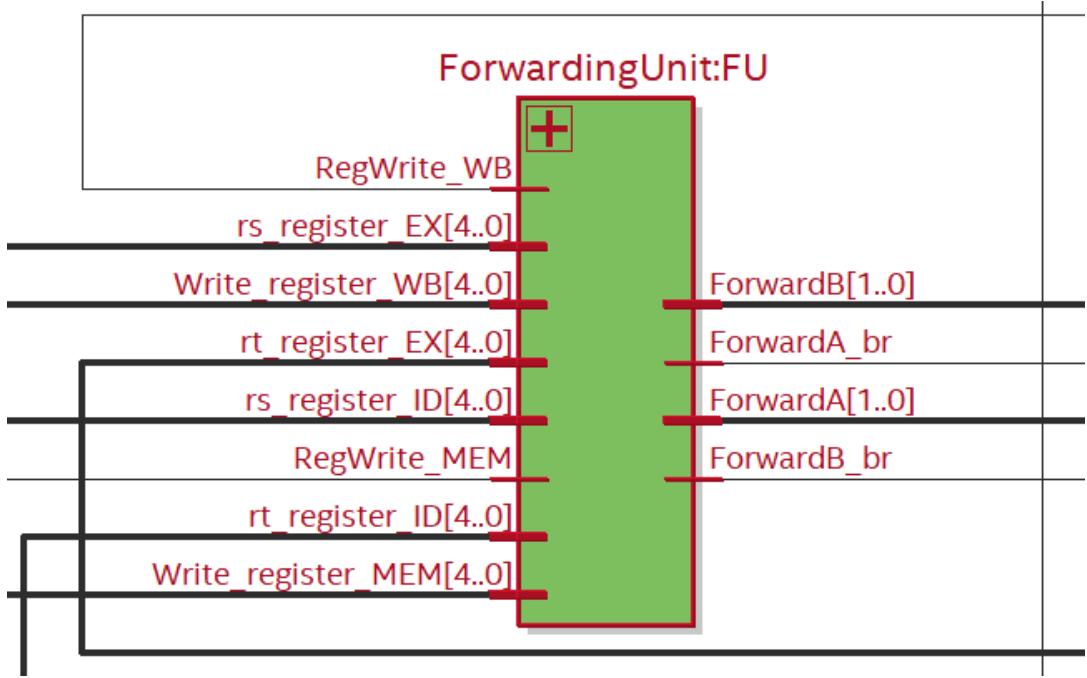




Forwarding:

מטרת הרכיב הינו להביא מידע מוקן משלב ה-MEM ושלב ה-WB אל ה-EXE וה-ID.

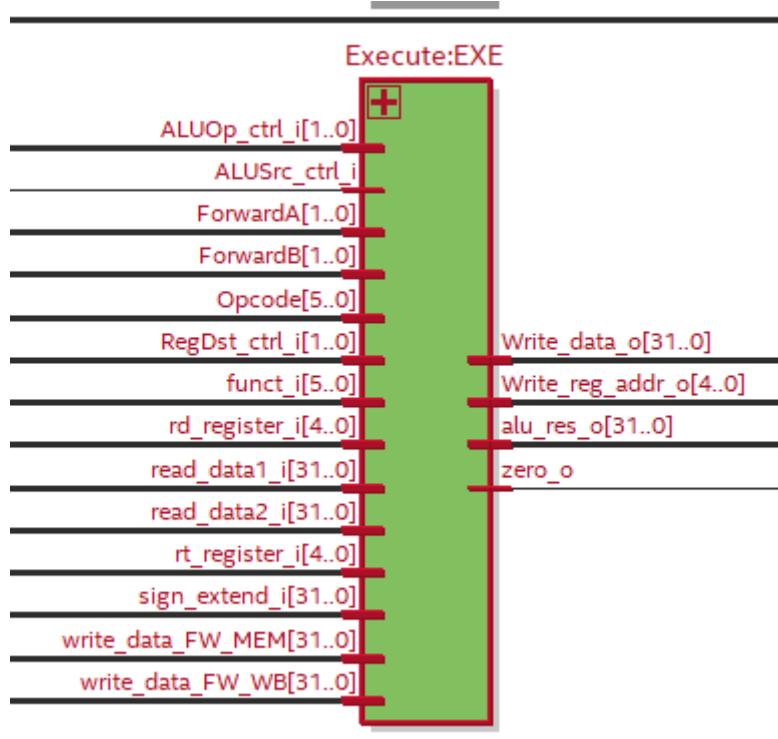
במקרה של פעולה ב-EXE דורשת אופרנדים שעוד לא הגיעו לRF אבל כבר כנ חושבו ונמצאים ב-Pipeline אז נחברם לשני הכניסות של ה-U ALU בעזרת MUX. בנוסף במקרה של Branch equal/not equal אז הרכיב עושה Forwarding אל ה-ID וזו בת מטרת הקטין את כמות השלבים שדורשות ההסתעפויות וכן בעצם גם להקטין את כמות ה-Flush אם אכן ההסתעפויות כן נלקחות. החלטה אם אכן לעשות Forward תלואה אם אחד מהרגיסטרים (שם האופרנדים) של פעולה שבديוק הגיעו לשלב ה-ID הוא רגיסטר המטרה של פעולה שנמצאות בשלבי ה-EXE וה-MEM.

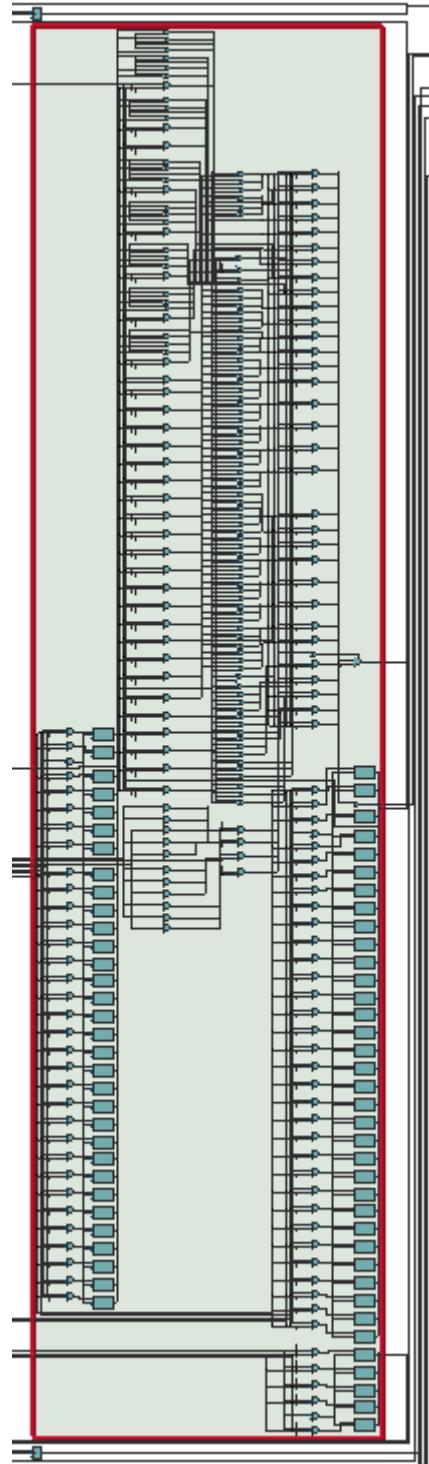


Execute:

במודול זה קורות הפעולות הארטמטיות במעבד ברכיב הALU, כדוגמא איזה ערך להכניס לרגיסטר או מה כתוב לזכרון.

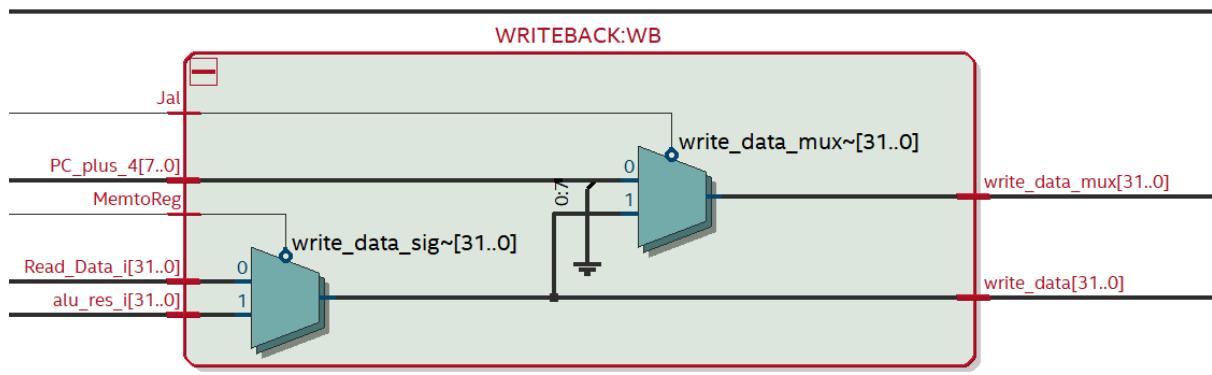
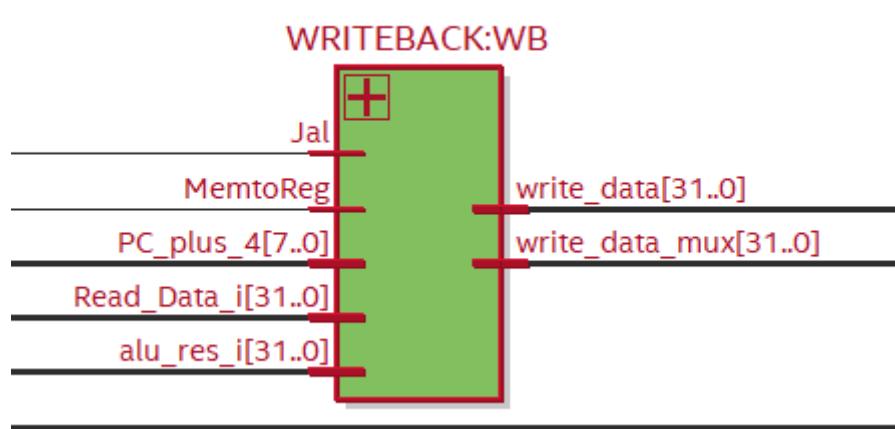
בנוסף בשלב זה קיים forwarding , עליו מפורט בהמשך.





Writeback:

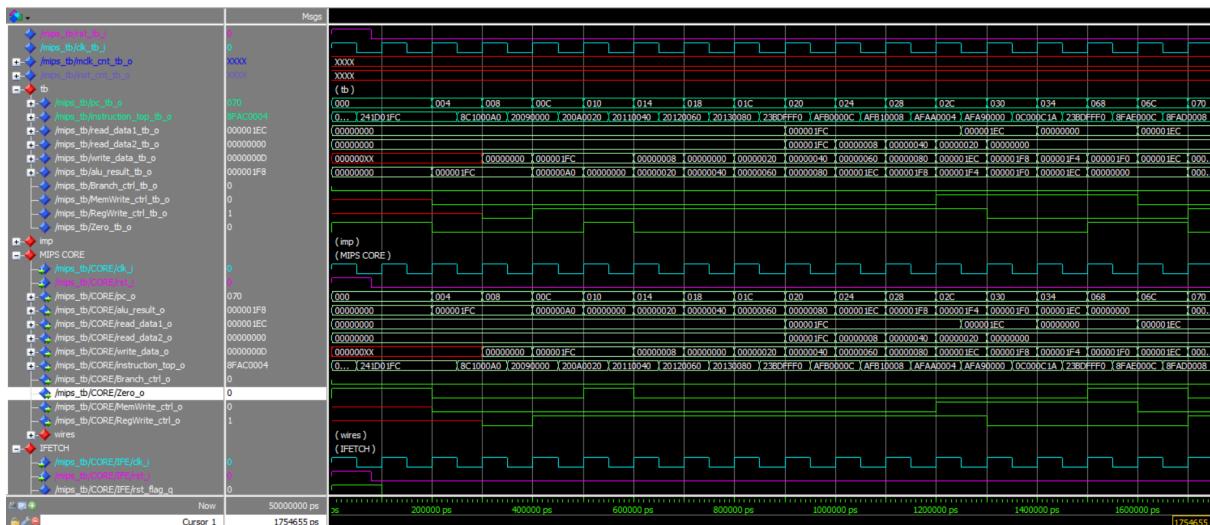
השלב האחרון ב-Pipeline, אחראי להעברת מידע ל-RF
ובורר איזה מידע להעברת (תוצאה של ה-ALU, מוצא
מהזיכרון או PC + 4 במקורה של JAL) ובורר ע"י קווים
בקשה שיוצאים מרכיב ה-Control.



:Model sim and Quartus simulation

ביצענו כמה סימולציות בכדי לבדוק את תקינות המערכת.
להלן 3 test שהועלה לנו כדוגמא:

Wave מה- חלק



תוצאת ה-DCM

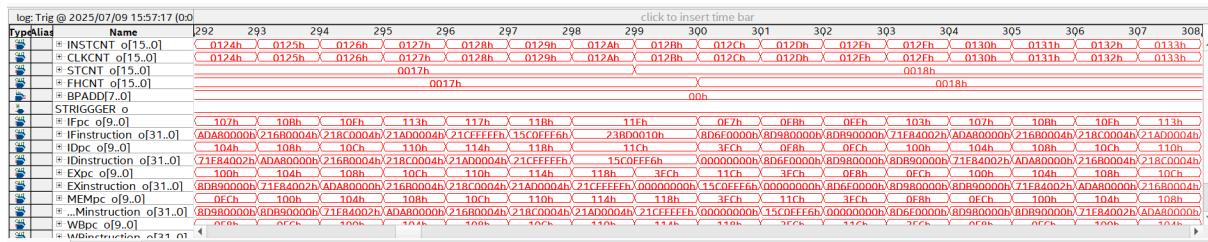
Memory Data - /mips_tb/CORE/G1/MEM/data_memory/MEMORY/m_mem_data_a - Default															
000000ff	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
000000f4	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
000000e9	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
000000de	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
000000d3	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
000000c8	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
000000bd	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
000000b2	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
000000a7	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0000009c	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000091	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000086	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000008	00000080	00000020			
0000007b	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000070	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000065	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0000005a	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0000004f	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000044	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
00000039	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000	00000000
0000002e	00000000	00000000	00000000	00000000	00000000	00000000	00000008	000002E8	00000292	0000023A	000001E0				
00000023	00000184	00000126	000000C6	00000064	FFFFF9AB	FFFFFA9	FFFFFA7	FFFFFA5	FFFFFA3	FFFFFA1	FFFFF9F				
00000018	FFFFFFFD	00000065	00000065	00000065	00000065	00000065	00000065	00000065	00000065	00000065	0000005D	0000005E			
0000000d	0000005F	00000060	00000061	00000062	00000063	00000064	00000008	00000007	00000006	00000005	00000004				
00000002	00000003	00000002	00000001												

.FPGA ביצעו טסטים ב-Quartus שרכז על ה-DTCM
להלן התוצאות ב-DTCM:

test3:

Instance 1: DTCM															
000000	00	00	00	01	00	00	00	02	00	00	00	03	00	00	00
000008	00	00	00	64	00	00	00	63	00	00	00	62	00	00	00
000010	00	00	00	65	00	00	00	65	00	00	00	65	00	00	00
000018	FF	FF	FF	9D	FF	FF	FF	9F	FF	FF	FF	A1	FF	FF	FF
000020	00	00	00	64	00	00	00	C6	00	00	01	26	00	00	01
000028	00	00	00	08	00	00	00	00	00	00	00	00	00	00	00
000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000038	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000048	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000058	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

SIGNAL TAP :



מעבר מקובץ C לקובץ אסמלים והטסט שלו:

קיבלנו קובץ C שמחבר ערכיהם של 2 מטריצות למטריצה שלישית, תרגמנו אותו לקובץ אסמבלי וביצענו עליו טסט ב Quartus .

קובץ C

```
#define M 4

void addMats(int Mat1[M][M], int Mat2[M][M], int resMat[M][M]){
    define it yourself ...
}

void main(){ //int=32bit
    int Mat1[M][M]={{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
    int Mat2[M][M]={{13,14,15,16},{9,10,11,12},{5,6,7,8},{1,2,3,4}};
    int resMat[M][M];

    addMats(Mat1,Mat2,resMat); // resMat = Mat1 + Mat2
}
```

קובץ אסמבלי:

```
.data

arr1: .word 1 ,2 ,3 ,4 ,5 ,6 ,7 ,8 ,9,10,11, 12, 13, 14, 15, 16,
arr2: .word 13, 14 ,15, 16, 9, 10, 11 ,12, 5 ,6, 7 ,8, 1, 2 ,3 ,4,
res1: .space 64 # SIZE*4=32[Byte] - ADD result array
SIZE: .word 16

.text
.globl main

main:
    li $sp,0x01FC      # stack initial address is 200
    lw $s0,SIZE($0)     # s0 = SIZE
    la $t1,arr1          # t1 points to arr1
    la $t2,arr2          # t2 points to arr2
    la $s1,res1          # s1 points to res
loop:
    addi $sp,$sp,-16
    sw $s0,12($sp)      # push SIZE
    sw $s1,8($sp)        # push res1 pointer
    sw $t2,4($sp)        # push arr2 pointer
    sw $t1,0($sp)        # push arr1 pointer
    jal mat_add

finish: beq $zero,$zero,finish
```

פתרונות:

Instance 1: DTCM															
000000	00	00	00	01	00	00	00	02	00	00	00	03	00	00	00
000006	00	00	00	07	00	00	00	08	00	00	00	09	00	00	00
00000c	00	00	00	0D	00	00	00	0E	00	00	00	0F	00	00	00
000012	00	00	00	0F	00	00	00	10	00	00	00	09	00	00	00
000018	00	00	00	05	00	00	00	06	00	00	00	07	00	00	00
00001e	00	00	00	03	00	00	00	04	00	00	00	0E	00	00	00
000024	00	00	00	0E	00	00	00	10	00	00	00	12	00	00	00
00002a	00	00	00	12	00	00	00	14	00	00	00	0E	00	00	00

