מעבדת ארכיטקטורת מעבדים מתקדמת ומאיצי חומרה

דו״ח מעבדה 5

מיכאל קדיק 316645415
עמית ניסני 205664345

תיאור המעבדה

במעבדה זו התבקשנו לייצר Scalar Pipelined MIPS CPU. לצורך כך, קיבלנו מעבד Single Cycle פשוט, הרחבנו את סט הפקודות בו הוא יכול לתמוך בהתאם לטבלה של סט הפקודות שנדרשנו להוסיף. לאחר מכן, חצצנו באמצעות רגיסטרים בין השלבים השונים: Fetch, Decode, Execute, Memory, Write Back והוספנו יחידות לטיפול בתלויות: Hazard Detection Unit, Forwarding Unit. בכדי להגיע לתוצאה הסופית, פעלנו עפ"י השלבים המתוארים במטלה. ניתן לראות את הארכיטקטורה של המעבד אותה מימשנו באיורים הבאים.

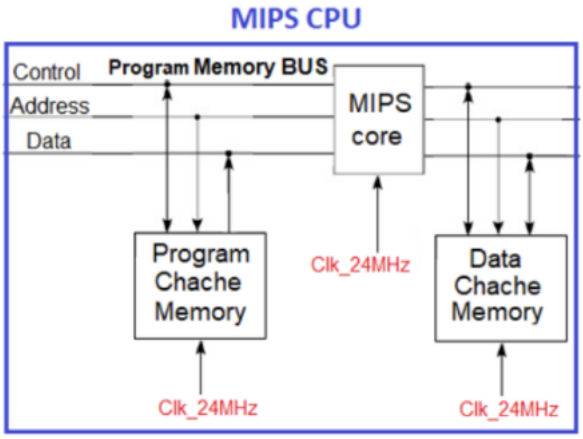

Figure 1. System architecture

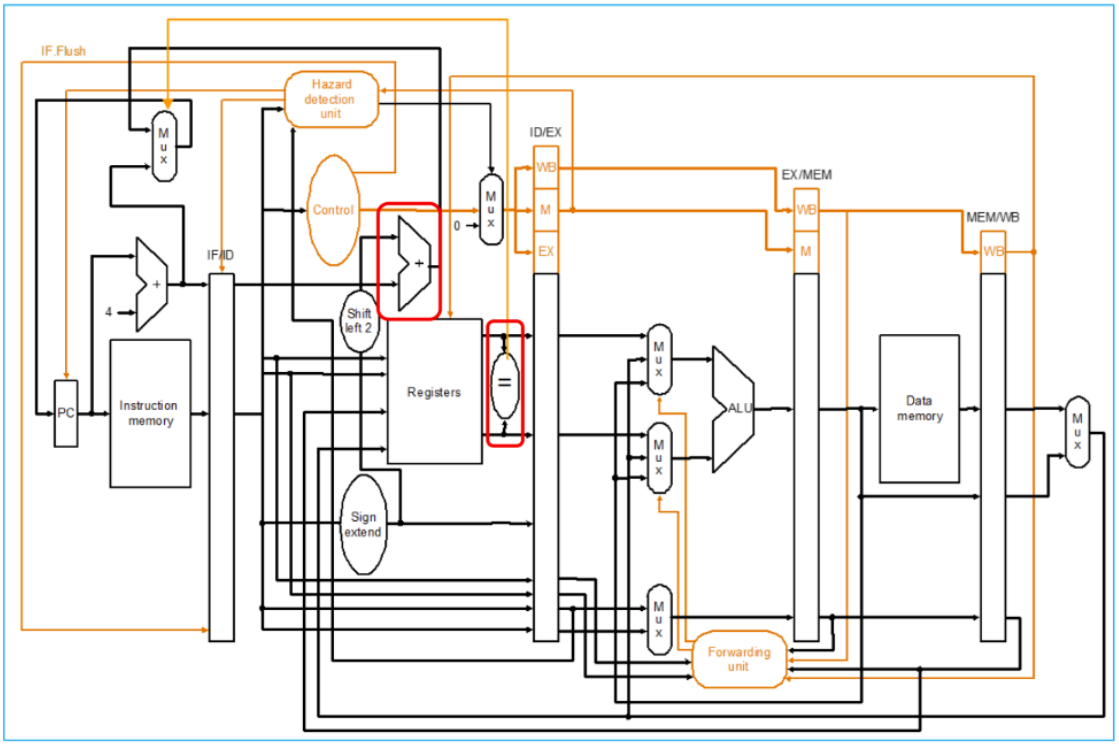

Figure 2. five stages pipelined MIPS architecture with forwarding and single delay slot

כמו כן, הוספנו במעטפת החיצונית יציאות בכדי שנוכל לעקוב אחרי התוכנית באופן הבא:

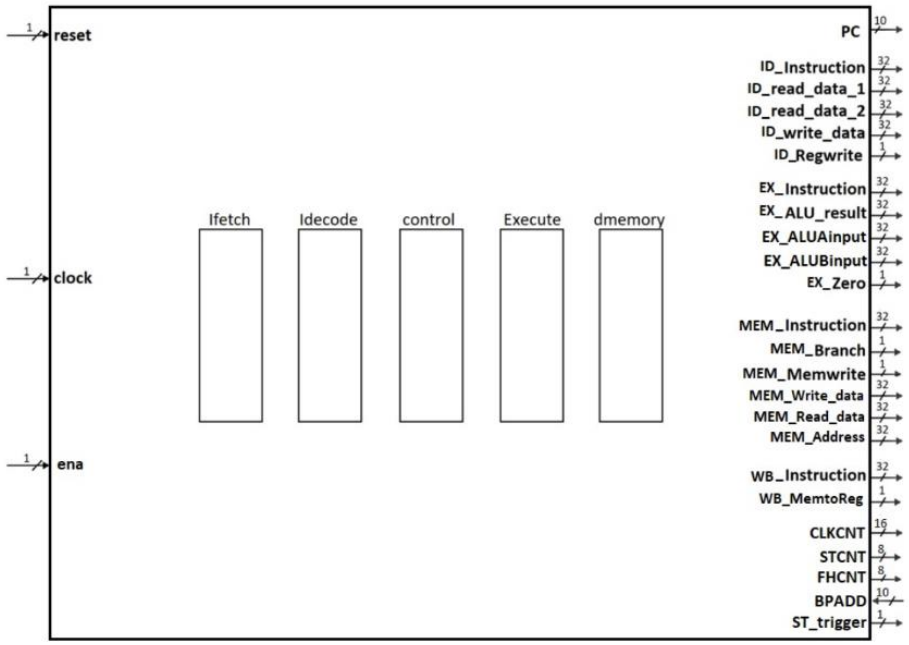

Figure 3. Pipelined MIPS architecture top entity

בכדי ליצור את המעבד המצוין למעלה, השתמשנו ברכיבי חומרה עליהם נפרט באיורים הבאים.

MIPS

רכיב זה הינו ה- top level entity שלנו ומטרתו לסנכרן בין כל רכיבי החומרה המפורטים בהמשך ולשמש כמעטפת חיצונית גם כן.

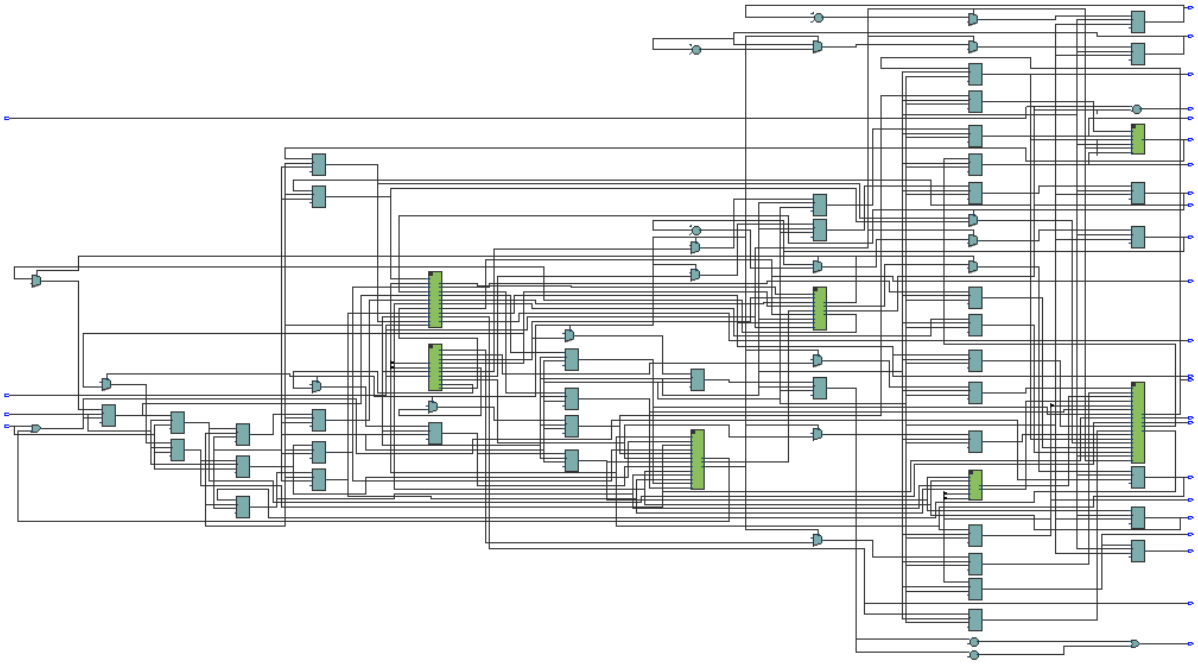

Figure 4. RTL Viewer of MIPS

Timing Models	Final
Logic utilization (in ALMs)	2,150 / 41,910 (5 %)
Total registers	3487
Total pins	54 / 499 (11 %)
Total virtual pins	421
Total block memory bits	552,960 / 5,662,720 (10 %)
Total DSP Blocks	2 / 112 (2 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI PMA TX Serializers	0 / 9 (0 %)
Total PLLs	0 / 15 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 5. Logic Usage

IFETCH

תפקידו של רכיב זה הוא להביא את הפקודה שיש לבצע ולקדם את ה-pc לפקודה הבאה (שאינה בהכרח הפקודה העוקבת, כמו למשל אם יש jump).

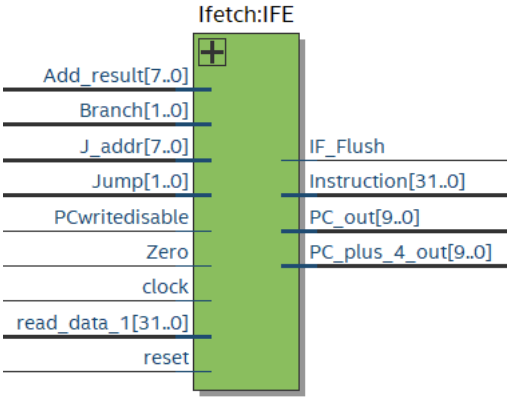

Figure 6. Graphical description of IFETCH

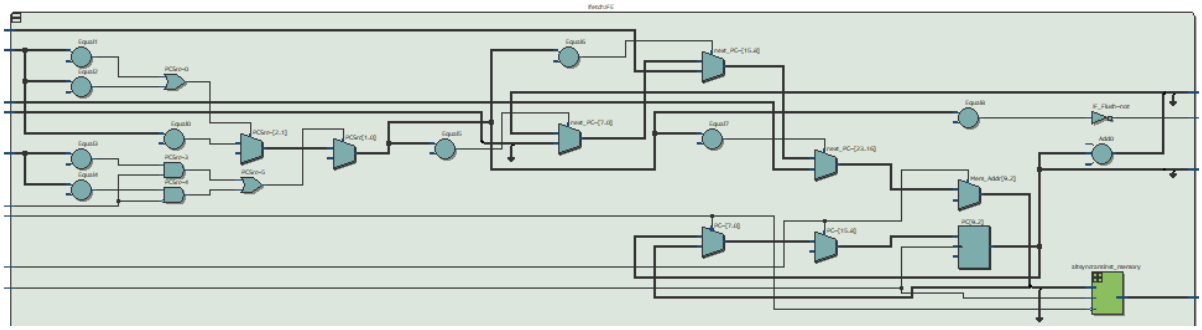

Figure 7. RTL Viewer of IFETCH

Name	Direction	Size	Functionality
clock	IN	1	שעון
reset	IN	1	אתחול
Add_result	IN	8	כתובת הסתעפות
read_data_1	IN	32	משמש לפקודת קפיצה
J_addr	IN	8	כותבת קפיצה
Branch	IN	2	בקרה
Jump	IN	2	בקרה
Zero	IN	1	בקרה
PCwritedisable	IN	1	משמש לעיכוב הפקודות
Instruction	OUT	32	פקודה לביצוע
PC_plus_4_out	OUT	10	כתובת של פקודה הבאה
PC_out	OUT	10	כתובת של פקודה נוכחית
IF_Flush	OUT	1	בקרה לריקון הרגיסטר

Table 1. Port Table of IFETCH

IDECODE

תפקידו של רכיב זה הוא לפענח את הפקודה שיש לבצע, בנוסף ברכיב זה נמצא הregister file ולכן מתבצעת בו גם כתיבה לרגיסטרים.

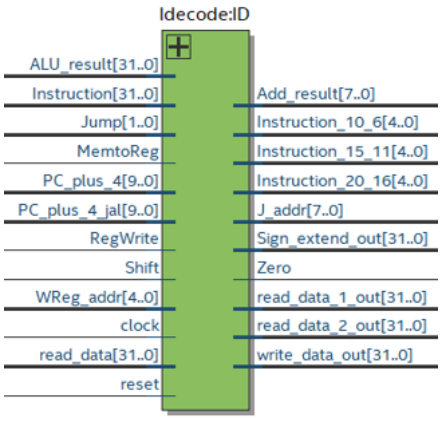

Figure 8. Graphical description of IDECODE

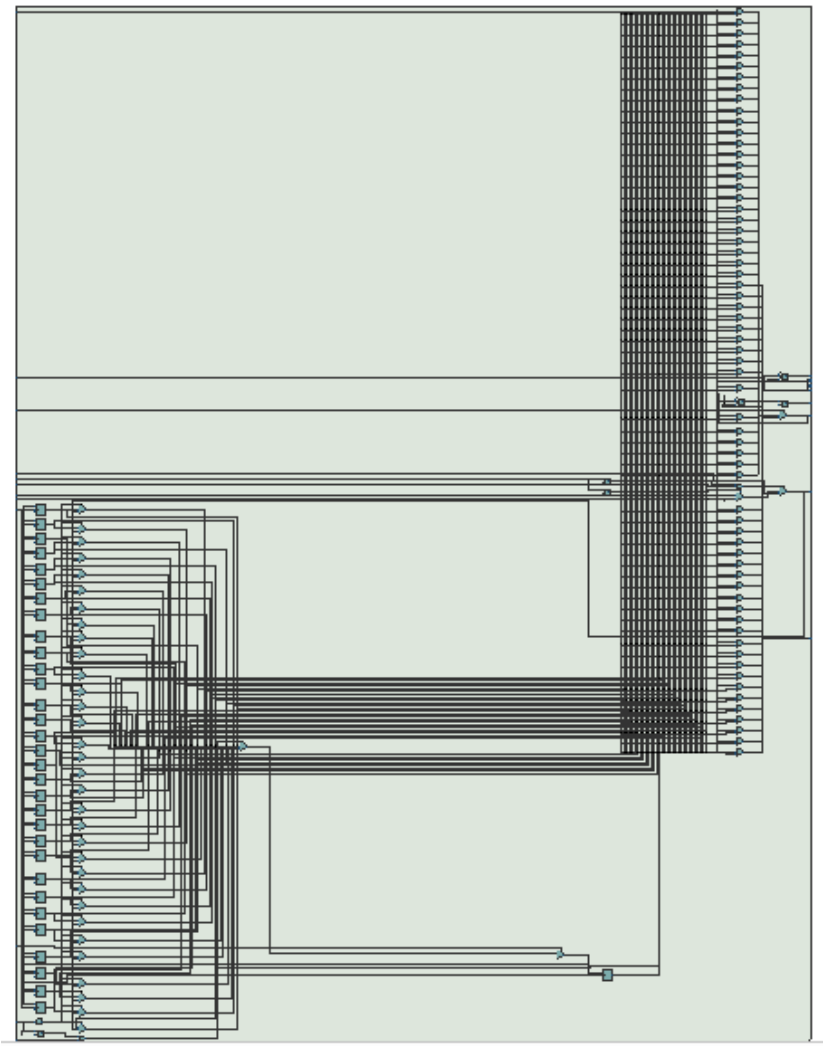

Figure 9. RTL Viewer of IFETCH

Name	Direction	Size	Functionality
clock	IN	1	שעון
reset	IN	1	אתחול
Instruction	IN	32	הפקודה הנוכחית לפענוח
read_data	IN	32	המידע בכתיבה לרגיסטרים
ALU_result	IN	32	תוצאת הפעולה ב- ALU
RegWrite	IN	1	בקרה
MemtoReg	IN	1	בקרה
Jump	IN	2	בקרה
Shift	IN	1	בקרה
PC_plus_4_jal	IN	10	המידע לכתיבה לרגיסטר במידה ויש jal
PC_plus_4	IN	10	כתובת של הפקודה הבאה לביצוע
WReg_addr	IN	5	כתובת המעידה לאיזה רגיסטר יש לכתוב
read_data_1_out	OUT	32	המידע מרגיסטר 1
read_data_2_out	OUT	32	המידע מרגיסטר 2
write_data_out	OUT	32	המידע שנכתב לרגיסטרים
Sign_extend_out	OUT	32	המידע בשדה הimmidiate מורחב
Instruction_20_16	OUT	5	ביטים 16-20 בפקודה
Instruction_15_11	OUT	5	ביטים 11-15 בפקודה
Instruction_10_6	OUT	5	ביטים 6-10 בפקודה
Add_result	OUT	8	כתובת הסתעפות
Zero	OUT	1	בקרה
J_addr	OUT	8	כתובת קפיצה

Table 2. Port Table of IDECODE

CONTROL

תפקידו של רכיב זה הוא לייצר את קווי הבקרה בהתאם לפקודה הנוכחית.

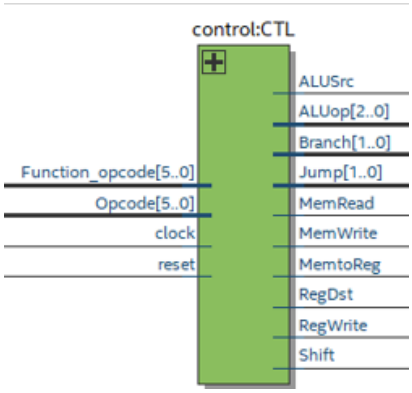

Figure 10. Graphical description of CONTROL

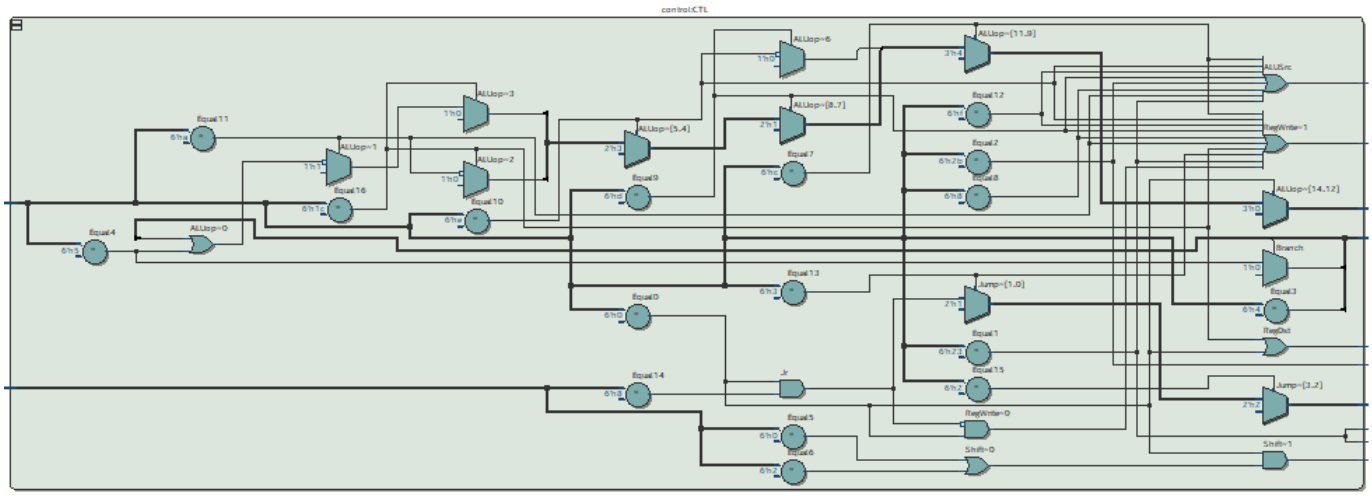

Figure 11. RTL Viewer of CONTROL

Name	Direction	Size	Functionality
clock	IN	1	שעון
reset	IN	1	אתחול
Opcode	IN	6	ביטים 26-31 בפקודה
Function_opcode	IN	6	ביטים 0-5 בפקודה
RegDst	OUT	1	קו בקרה לבחירת כתובת יעד הכתיבה לרגיסטר
ALUSrc	OUT	1	קו בקרה הבורר איזה מידע יכנס לכניסה B ב-ALU
MemtoReg	OUT	1	קו בקרה הבורר בין המידע שיכתב לתוך הרגיסטרים (מהזכרון או מה-ALU)
RegWrite	OUT	1	קו בקרה הבורר את כתובת הרגיסטר לכתיבה
MemRead	OUT	1	קו בקרה המבחין האם יש לקרוא מהזכרון
MemWrite	OUT	1	קו בקרה המבחין האם יש לכתוב לזכרון
Shift	OUT	1	קו בקרה המבחין האם יש פעולת shift
Jump	OUT	2	קו בקרה המבחין האם יש פעולת קפיצה
Branch	OUT	2	קו בקרה המבחין האם יש פעולת הסתעפות
ALUop	OUT	3	קו בקרה הבורר איזה פעולה יש לעשות ב-ALU

Table 3. Port Table of CONTROL

EXECUTE

תפקידו של רכיב זה הוא ביצוע הפעולות הנדרשות בכדי לבצע את הפקודה.

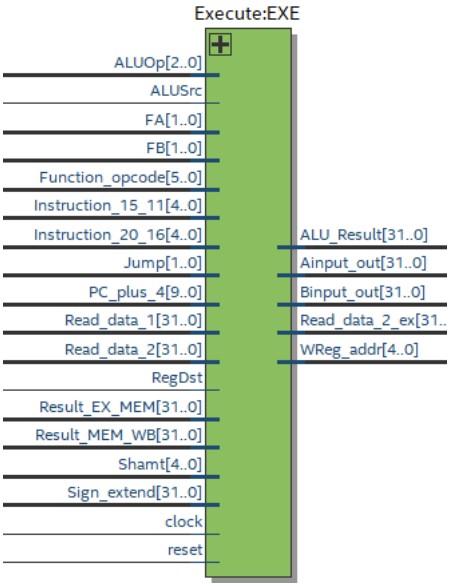

Figure 12. Graphical description of EXECUTE

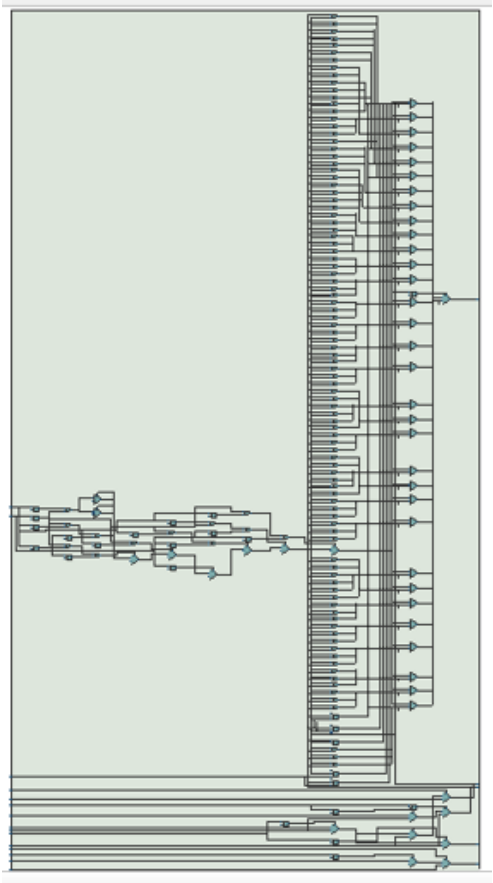

Figure 13. RTL Viewer of EXECUTE

Name	Direction	Size	Functionality
clock	IN	1	שעון
reset	IN	1	אתחול
Read_data_1	IN	32	מידע מרגיסטר 1
Read_data_2	IN	32	מידע מרגיסטר 2
Sign_extend	IN	32	מידע משדה הimmidiate מורחב
Function_opcode	IN	6	ביטים 0-5 של הפקודה
ALUSrc	IN	1	בקרה
FA	IN	2	בקרה
FB	IN	2	בקרה
ALUOp	IN	3	בקרה
Result_EX_MEM	IN	32	מידע מרגיסטר בהמשך למטרת forwarding
Result_MEM_WB	IN	32	מידע מרגיסטר בהמשך למטרת forwarding
PC_plus_4	IN	10	כתובת של הפקודה הבאה לביצוע
Jump	IN	2	בקרה
RegDst	IN	1	בקרה
Instruction_20_16	IN	5	ביטים 16-20 בפקודה
Instruction_15_11	IN	5	ביטים 11-15 בפקודה
Shamt	IN	5	ביטים 6-10 בפקודה
Ainput_out	OUT	32	מידע שנכנס לכניסה A ב-ALU
Binput_out	OUT	32	מידע שנכנס לכניסה B ב-ALU
ALU_Result	OUT	32	תוצאת ה-ALU
Read_data_2_ex	OUT	32	מידע מרגיסטר 2
WReg_addr	OUT	5	כתובת כתיבה לרגיסטרים

Table 4. Port Table of EXECUTE

DMEMORY

תפקידו של רכיב זה הוא לבצע קריאה מהזכרון או כתיבה לזכרון.

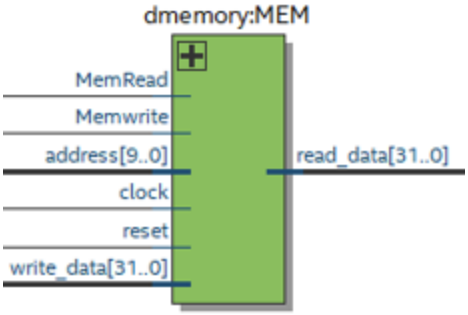

Figure 14. Graphical description of DMEMORY

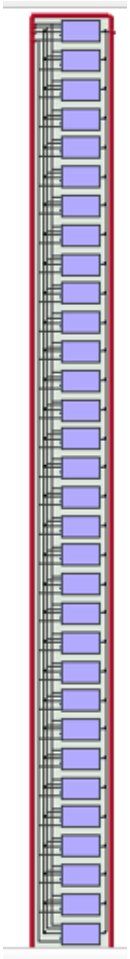

Figure 15. RTL Viewer of EXECUTE

Name	Direction	Size	Functionality
clock	IN	1	שעון
reset	IN	1	אתחול
address	IN	10	כתובת כתיבה לזכרון
write_data	IN	32	מידע שנכתב לזכרון
MemRead	IN	1	בקרה
Memwrite	IN	1	בקרה
read_data	OUT	32	מידע שיוצא מהזכרון

Table 5. Port Table of DMEMORY

HDU

תפקידו של רכיב זה הוא לזהות תלויות בין פקודות ולבצע השהיה במקרה הצורך.

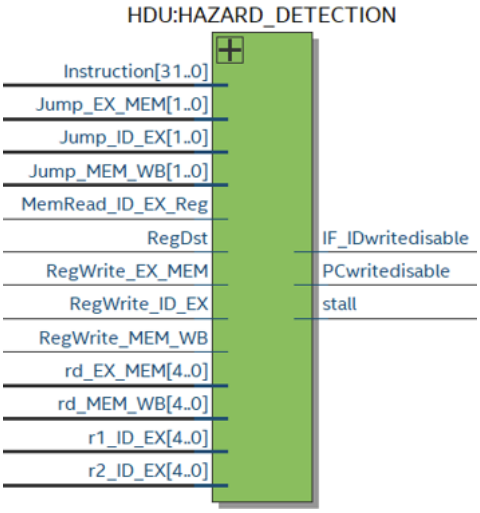

Figure 16. Graphical description of HDU

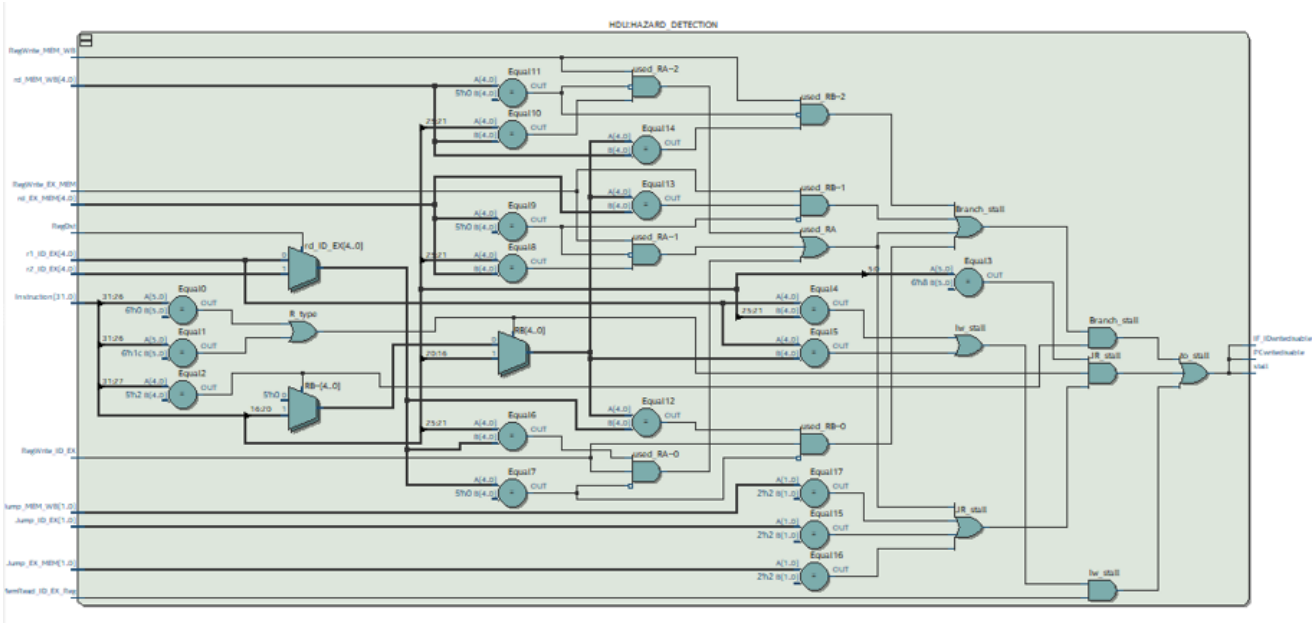

Figure 17. RTL Viewer of HDU

Name	Direction	Size	Functionality
Instruction	IN	32	פקודה לביצוע
r1_ID_EX	IN	5	רגיסטר 1
r2_ID_EX	IN	5	רגיסטר 2
rd_EX_MEM	IN	5	רגיסטר המטרה של פקודה קודמת שנמצאת ב-EXECUTE
rd_MEM_WB	IN	5	רגיסטר המטרה של פקודה קודמת שנמצאת ב-WB
Jump_ID_EX	IN	2	בקרה
Jump_EX_MEM	IN	2	בקרה
Jump_MEM_WB	IN	2	בקרה
RegWrite_ID_EX	IN	1	בקרה
RegWrite_EX_MEM	IN	1	בקרה
RegWrite_MEM_WB	IN	1	בקרה
RegDst	IN	1	בקרה
MemRead_ID_EX_Reg	IN	1	בקרה
PCwritedisable	OUT	1	אות בקרה שמתפקידו לעכב את ה- PC
IF_IDwritedisable	OUT	1	אות בקרה שתפקידו לעכב את רגיסטר IF_ID
stall	OUT	1	אות בקרה שתפקידו לעכב את המעבד

Table 6. Port Table of HDU

FU

תפקידו של רכיב זה הוא לזהות תלויות שניתן לפתור ללא השהיה ובמקרה הצורך להעביר את המידע משלב מתקדם יותר אל שלב ה-Execute.

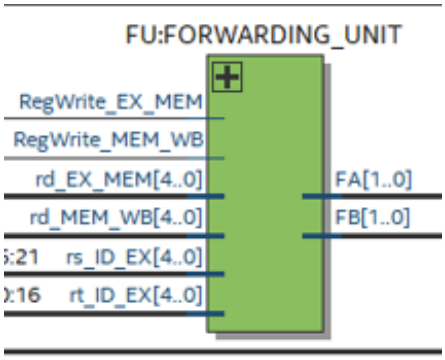

Figure 18. Graphical description of FU

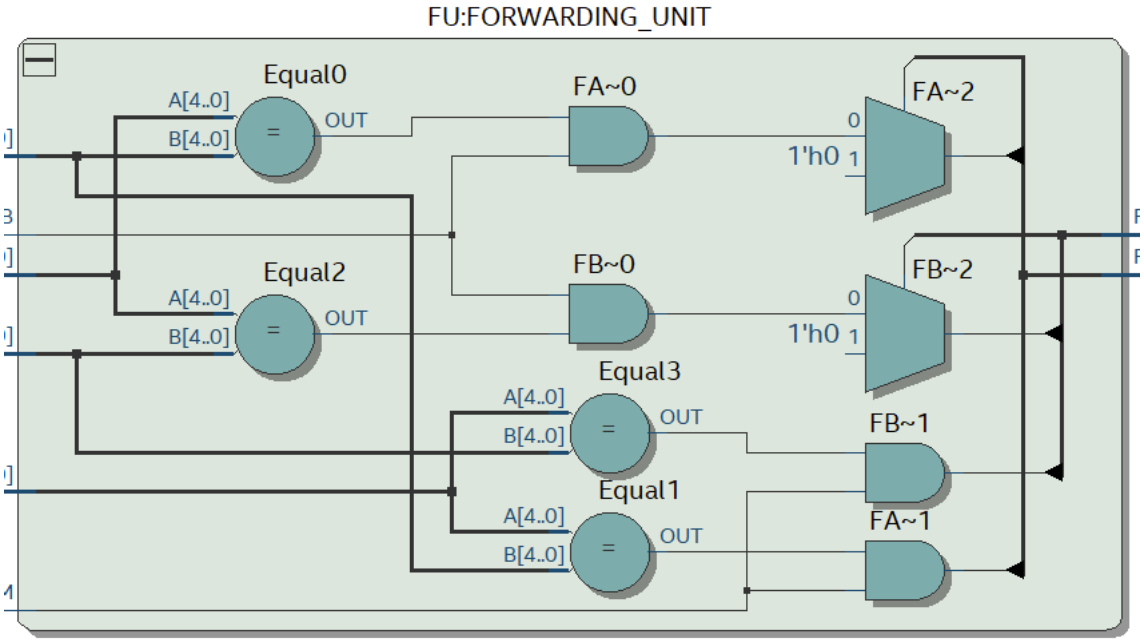

Figure 18. RTL Viewer of FU

Name	Direction	Size	Functionality
rs_ID_EX	IN	5	רגיסטר rs
rt_ID_EX	IN	5	רגיסטר rd
rd_EX_MEM	IN	5	רגיסטר מטרה משלב ה-MEM
rd_MEM_WB	IN	5	רגיסטר מטרה משלב ה-WB
RegWrite_EX_MEM	IN	1	בקרה
RegWrite_MEM_WB	IN	1	בקרה
FA	OUT	2	קו בקרה הבורר מה יכנס לכניסה A ב- ALU
FB	OUT	2	קו בקרה בורר מה יכנס לכניסה B ב-ALU

Table 7. Port Table of FU

כמו כן, בדקנו את ה-fmax ואת המסלול הארוך ביותר עבור מימוש ה single cycle ועבור המימוש הסופי. ניתן לראות אותם באיורים הבאים.

Fmax	Restricted Fmax	Clock Name	Note
39.48 MHz	39.48 MHz	clock	

Figure 19. Fmax of the Single Cycle MIPS

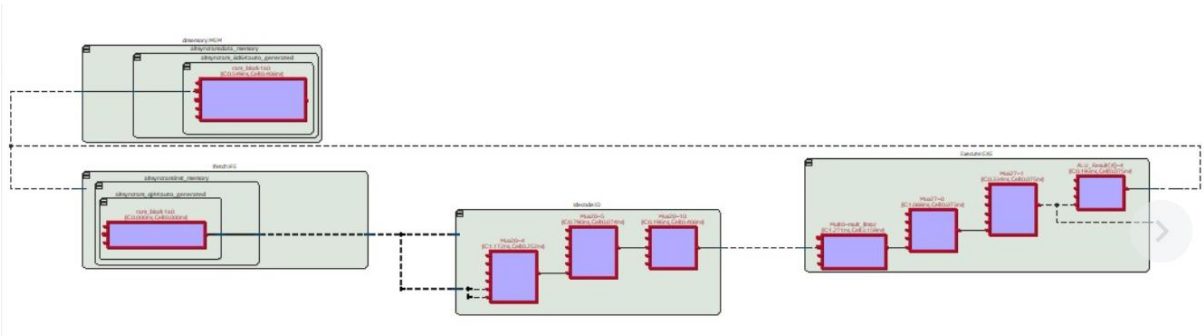

Figure 20. Critical Path of the Single Cycle MIPS

ניתן לראות כי המסלול הקריטי עובר בכל הרכיבים, כצפוי ממעבד Single Cycle.

Fmax	Restricted Fmax	Clock Name	Note
47.22 MHz	47.22 MHz	clock	

Figure 21. Fmax of the Pipelined MIPS

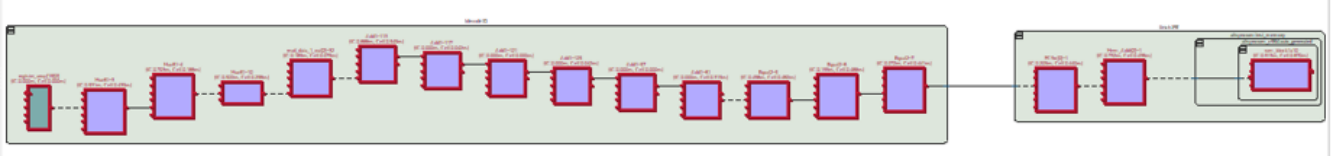

Figure 22. Critical Path of the Pipelined MIPS

ניתן להבחין כי המסלול הקריטי מתחיל בשלב ה-DECODE ונגמר בשלב ה-FETCH. דבר זה נובע מפקודות ההסתעפות בהן מתבצעת החלטה בשלב ה-FETCH על סמך חישובים שמתבצעים בDECODE. זוהי הפעולה היחידה שאין בה חציצה בין השלבים ולכן הגיוני שזה יהיה המסלול הארוך ביותר.

בכדי לבדוק את נכונות המימוש שלנו ושאכן המעבד מבצע את כל הפקודות כנדרש וכל רכיבי החומרה מבצעים את תפקידם כתבנו מספר תוכניות אסמבלי ובדקנו אותן באמצעות הModelsim.

```
.data

.text
main:
        add $v0,$at,$zero
        sub $a1,$a0,$v1
        ori $t1,$s4,8
        xori $v0,$at,6
        addi $a1,$zero,4
        and $a0,$v1,$a1
        or $a2,$a3,$t0
        xor $t1,$t2,$t3
        andi $v0,$s0,7
        srl $a0,$s5,4

DONE:   nop
        j DONE
```

Figure 23. First assembly program

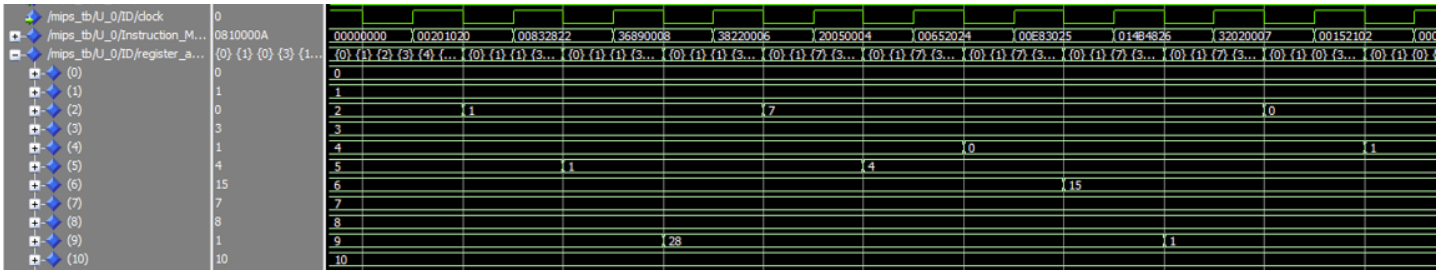

Figure 24. First assembly program results

בתוכנית זו בדקנו ששלל הפקודות האריתמטיות והלוגיות עובדות כראוי, ניתן לראות שהתוצאות שנכתבות לרגיסטרים אכן נכונות כמצופה.

כמו כן, הוספנו תלויות בין הפקודות השונות שניתנות לטיפול באמצעות forwarding. ניתן לראות זאת באיור הבא.

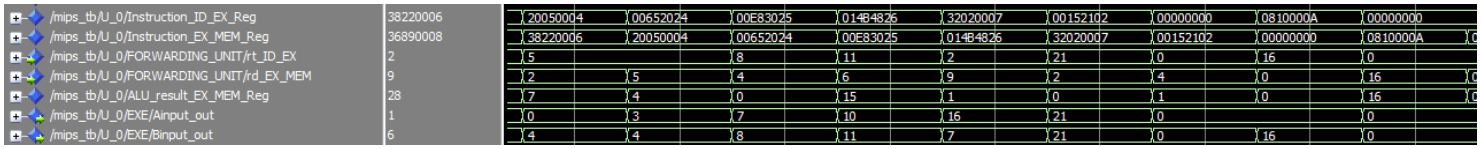

Figure 25. First assembly program forwarding

באיור זה ניתן להבחין בforwarding בשני מחזורי השעון הראשונים. ישנה פקודה (and) שצריכה לקרוא ערך ברגיסטר שפקודה קודמת (addi) כותבת אליו. ניתן לראות שהערך שנכנס לכניסה B של ה-ALU כאשר פקודת ה-and נמצאת ב-EXECUTE הוא התוצאה של פקודת ה-andi.

כמו כן, ניתן לראות את המעבר של פקודה בין הרגיסטרים השונים בפייפליין באיור הבא.

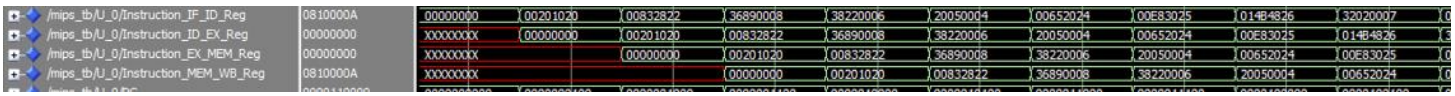

Figure 26. First assembly program instructions

בנוסף, כתבנו תוכנית אסמבלי נוספת שבה בדקנו שאכן ההסתעפויות ופקודות הshift עובדות כמו שצריך. כמו כן, בדקנו שאכן מתבצע stall ו-flush בהתאם לצורך. ניתן לראות אותה ואת התוצאות באיורים הבאים.

```
.data

.text
main:
        addi $v0,$at,2
        bne $v0,$a1,NotEQ
        srl $a1,$a2,3
NotEQ:  sll $a1,$a2,3

DONE:   nop
        j DONE
```

Figure 27. Second assembly program

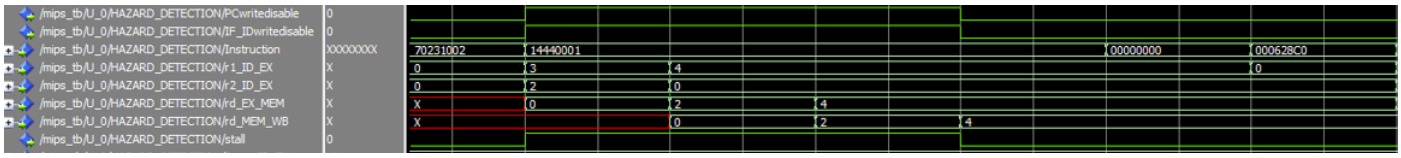

Figure 28. Second assembly program results

ניתן להבחין כי אכן כאשר נכנסת פקודת ההסתעפות אל ה-Hazard Detection Unit מצבצע stall ומיד לאחריו מתבצע flush מכיוון שצריך להסתעף. כמו כן, ניתן לראות שלאחר ה-flush התוכנית ממשיכה מהפקודה הנכונה.

כמו כן, התבקשנו לכתוב תוכנית אסמבלי שמבצעת חיבור מטריצות בהתאם לפסאודו-קוד הבא:

```
#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
}
```

Figure 29. Pseudo-code as given in the assignment

ניתן לראות את התוכנית שכתבנו ואת תמונת הזכרון המעיד על הצלחתה בסוף התוכנית באיורים הבאים.

```
.data
        Mat1: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16
        Mat2: .word 13, 14, 15, 16, 9, 10, 11, 12, 5, 6, 7, 8, 1, 2, 3, 4
        resMat: .space 64 #16 x 4

.text
main:
        addi $t0,$zero,16 # Loop index
        addi $t1,$zero,0 # Mat1 index
        addi $t2,$zero,64 # Mat2 index
        addi $t3,$zero,128 # resMat index

addMats:
        # Load the relevant values, add them and store them in resMat
        lw,$t5,0($t1)
        lw,$t6,0($t2)
        add $t7,$t5,$t6
        sw $t7,0($t3)

        #update indexes
        addi $t0,$t0,-1 # Loop index
        addi $t1,$t1,4  # Mat1 index
        addi $t2,$t2,4 # Mat2 index
        addi $t3,$t3,4 # resMat index

        # Check if loop ended
        bne $t0,$zero, addMats

DONE:   nop
        j DONE
```

Figure 30. Assembly program for matrix sum

255	0	0	0	0	0	0	0	0	0	0	0	0
243	0	0	0	0	0	0	0	0	0	0	0	0
231	0	0	0	0	0	0	0	0	0	0	0	0
219	0	0	0	0	0	0	0	0	0	0	0	0
207	0	0	0	0	0	0	0	0	0	0	0	0
195	0	0	0	0	0	0	0	0	0	0	0	0
183	0	0	0	0	0	0	0	0	0	0	0	0
171	0	0	0	0	0	0	0	0	0	0	0	0
159	0	0	0	0	0	0	0	0	0	0	0	0
147	0	0	0	0	0	0	0	0	0	0	0	0
135	0	0	0	0	0	0	0	0	0	0	0	0
123	0	0	0	0	0	0	0	0	0	0	0	0
111	0	0	0	0	0	0	0	0	0	0	0	0
99	0	0	0	0	0	0	0	0	0	0	0	0
87	0	0	0	0	0	0	0	0	0	0	0	0
75	0	0	0	0	0	0	0	0	0	0	0	0
63	0	0	0	0	0	0	0	0	0	0	0	0
51	0	0	0	0	20	18	16	14	20	18	16	14
39	20	18	16	14	20	18	16	14	4	3	2	1
27	8	7	6	5	12	11	10	9	16	15	14	13
15	16	15	14	13	12	11	10	9	8	7	6	5
3	4	3	2	1								

Figure 31. Data Memory at the end of the program

ניתן לראות שבסוף הריצה, שלושת המטריצות נשמרות בזכרון, שתי המטריצות הראשונות שמוגדרות מראש והמטריצה של הסכום שלהן. כמו כן, ניתן לראות כי מטריצת הסכום היא בעלת ערכים נכונים.

בנוסף, בדקנו את נכונות המימוש שלנו באמצעות הSignal Tap ב- Quartus. ניתן לראות את התוצאות באיור הבא.

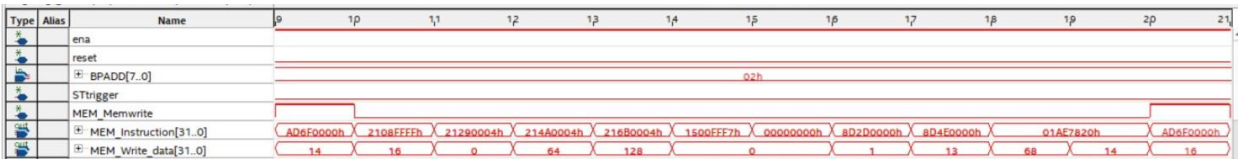

Figure 32. Signal Tap Verification

ניתן לראות שכאשר פקודת ה-sw מגיעה אל שלב ה-MEMORY, אכן יוצא הערך הנכון ב – MEM_Write_data, כמו כן עולה קו הבקרה MEM_Memwrite ולכן אנו מניחים שהערך גם נכנס לזכרון. מכיוון שאין לנו דרך לצפות בזכרון ב-Quartus לא יכלנו לוודא זאת.

בנוסף, הוספנו רגיסטרים שסופרים את כמות מחזורי השעון, stall ו- flush שהיו במהלך התוכנית. ניתן לראות את הערכים שלהם בסוף התוכנית באיור הבא.

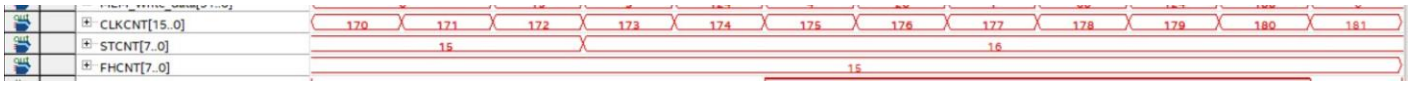

Figure 33. Signal Tap counters values

ניתן לראות את ערכי הרגיסטרים בסוף התוכנית: מספר מחזורי השעון הוא 180, מספר ה-stall הוא 16 ומספר ה-flush הוא 15. בעזרת נתונים אלו, חישבנו את ה-IPC באופן הבא:

$$IPC = \frac{CLKCNT - (STCNT + 4 + FHCNT * dept)}{CLKCNT} = \frac{180 - (16 + 4 + 15 * 1)}{180} = 0.806$$