ארכיטקטורת CPU

מעבדה 3

מגישים: אייר גלעד 300309937

דודו אבו 038109484

תוכן עניינים:

מטרת המעבדה

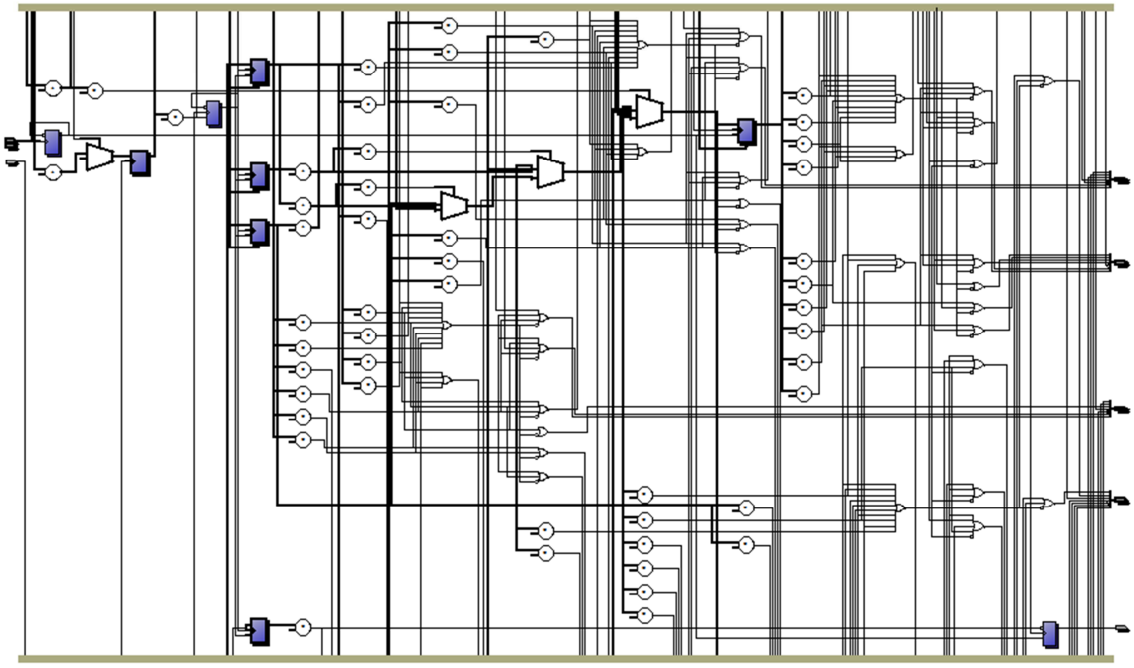
- Design, synthesize and analyze a simple MIPS compatible CPU.
- Understanding in FPGA memory structure

הטיימר

עקרון הפעולה של הטיימר פשוט ומבוסס על הרעיון של הטיימר מבקר ה MSP.

אנו סופרים עליות שעון ומשווים עם ערך שהוכנס לרגיסטר יעודי. בהגעת ה counter לערך הרגיסטר מופעל קו בקרה שגורם לאינטרפט.

כאשר מופעל קו האינטרפט מוכנס לרגיסטר ה PC כתובת הלייבל של רוטינת האינטרפט ותכנית האסמבלי ממשיכה משם.



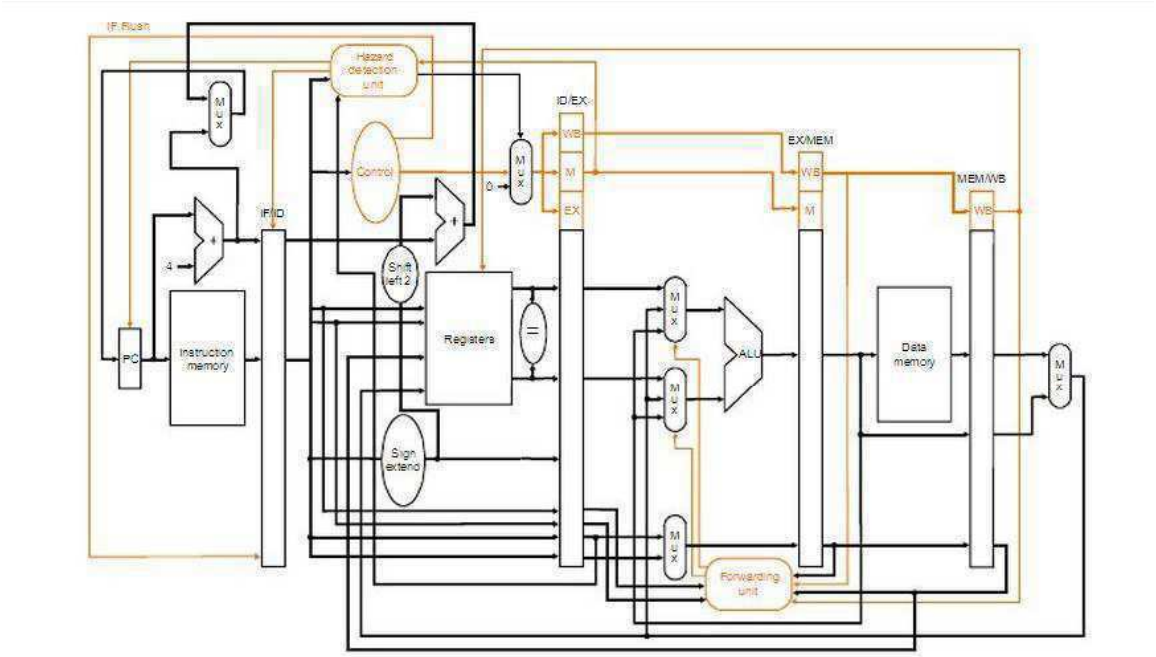
איור 0: תרשים הטיימר

סכמה כללית של המעבד והסבר:

המעבד אותו יצרנו מבוסס Pipelined MIPS ובעלת 4 שלבים:

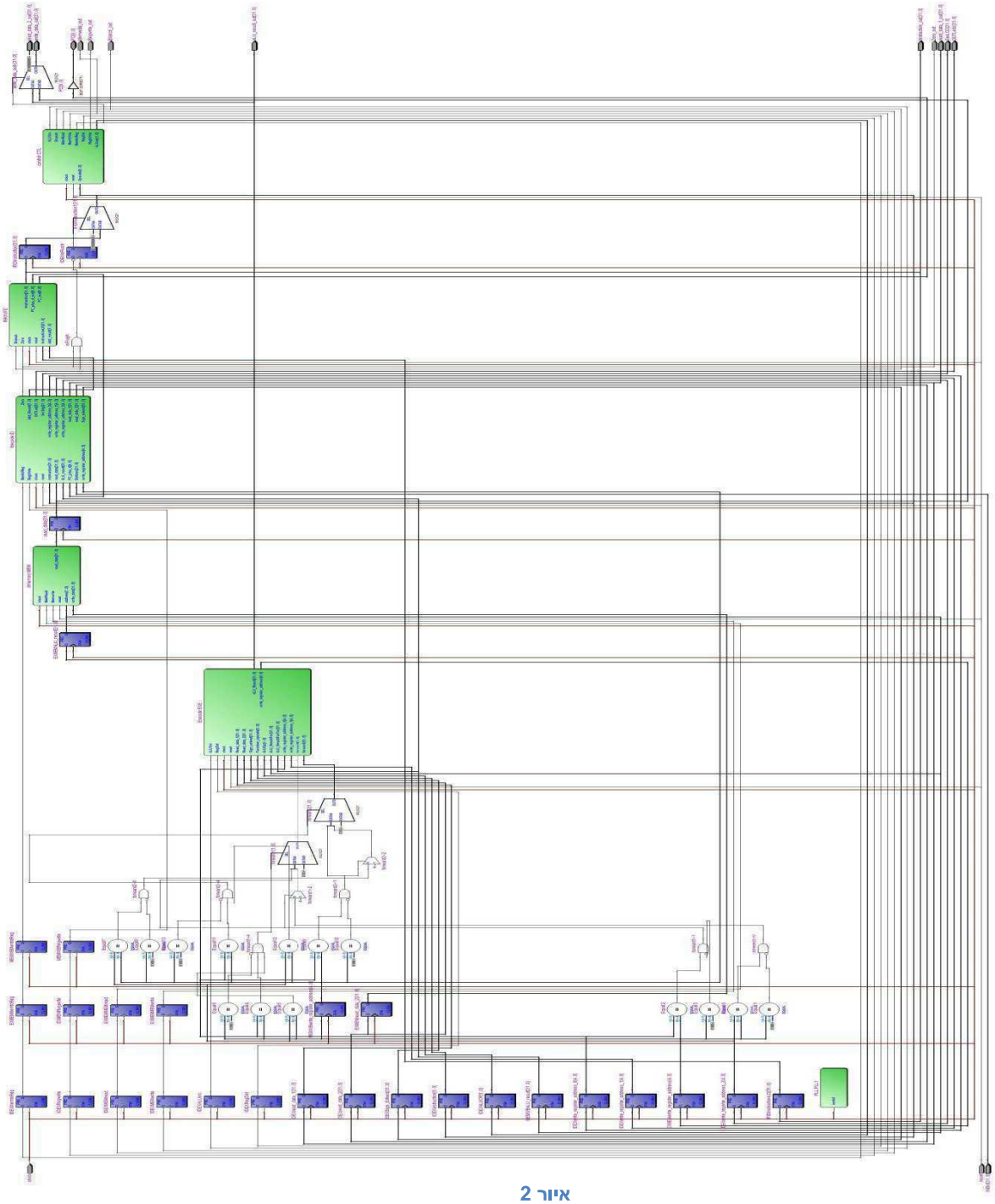
- Instruction Fetch
- Instruction Decode
- Execute
- Memory and Write Back

שלבים אלו מהווים את המודלים העיקריים ועליהם נפרט בהמשך. שלבים אלו מופרדים ביניהם על ידי רגיסטרים. בכל עלית שעון כל שלב מעביר לשלב הבא בתור את תוצאותיו, כך, ברגע נתון בכל אחד מהשלבים נמצאת פקודה אחרת, כאשר אין פגיעה באף אחד מהנתונים של השלבים אודות לרגיסטרים שחוצצים בין השלבים.
כך למעשה מתבצעת עבודה במקביל של כל השלבים ומתאפשרת עבודה עם תדר שעון גבוה יותר.
עם זאת, לתצורה זו של Pipeline מספר בעיות שצצות עקב העבודה במקביל, ועליהן נפרט בהמשך.
באיור הבא ניתן לראות תיאור סכמתי של המעבד:



איור 1

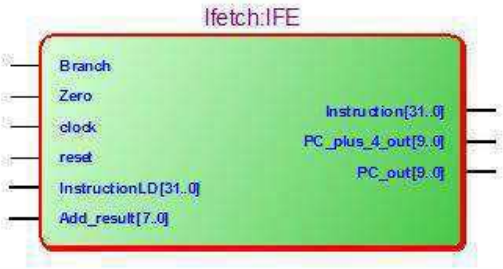
Top level block review diagram



איור 2

שלב instruction fatch :

שלב זה הוא השלב הראשון מתוך ארבעת השלבים של המעבד ותפקידו הבאת פקודות מזיכרון המערכת.

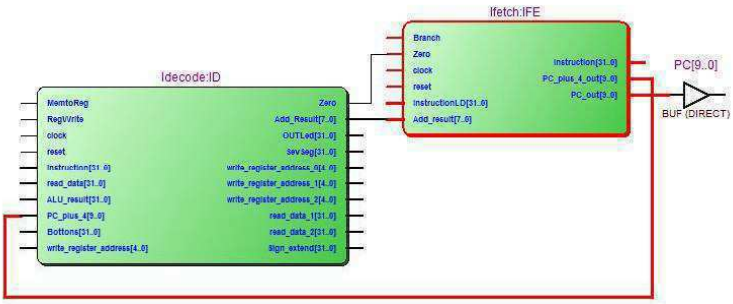


איור 3

Port map:

```
PORT( Instruction   : OUT  STD_LOGIC_VECTOR( 31 DOWNTO 0 );
SIGNAL InstructionLD              : in   STD_LOGIC_VECTOR( 31 DOWNTO 0 );
PC_plus_4_out               : OUT        STD_LOGIC_VECTOR( 9 DOWNTO 0 );
Add_result                  : IN   STD_LOGIC_VECTOR( 7 DOWNTO 0 );
Branch                            : IN   STD_LOGIC;
Zero                        : IN   STD_LOGIC;
PC_out                            : OUT  STD_LOGIC_VECTOR( 9 DOWNTO 0 );
clock,reset          : IN   STD_LOGIC );
```

מסלול קריטי של הרכיב :



איור 4

שלב הInstruction Decode

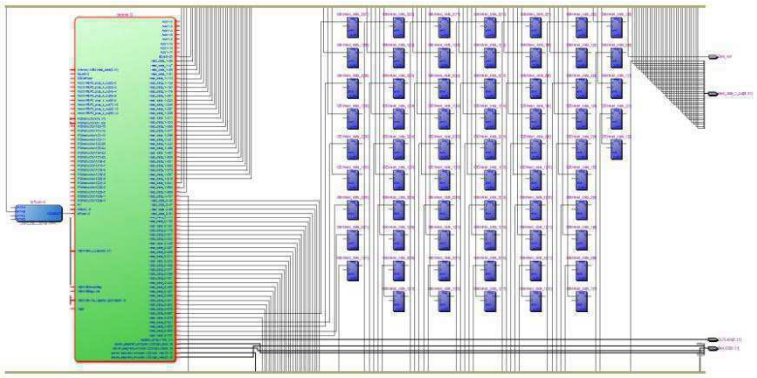
שלב זה הוא השלב השני מתוך ארבעת השלבים של המעבד.
שלב זה הינו אולי השלב המשמעותי ביותר מתוך ארבעת שלבי ה Pipeline של המעבד.
תפקיד שלב זה הינו פענוח הפקודה שהתקבלה משלב ה IF הקודם, העברת קווי הבקרה השונים לרכיבי ושלבי המערכת השונים, והעברת ערכי הרגיסטרים המתאימים לשלב הבא, בנוסף שלב זה מכיל את יחידת הcontrol עליה נפרט בנפרד.

איור 5

Port map:

```
PORT(  read_data_1          : OUT  STD_LOGIC_VECTOR( 31 DOWNTO 0 );
read_data_2          : OUT  STD_LOGIC_VECTOR( 31 DOWNTO 0 );
Instruction          : IN   STD_LOGIC_VECTOR( 31 DOWNTO 0 );
read_data                   : IN   STD_LOGIC_VECTOR( 31 DOWNTO 0 );
ALU_result                  : IN   STD_LOGIC_VECTOR( 31 DOWNTO 0 );
RegWrite, MemtoReg   : IN   STD_LOGIC;
Add_Result    : OUT  STD_LOGIC_VECTOR( 7 DOWNTO 0 );--
Zero                 : OUT  STD_LOGIC;
PC_plus_4            : IN   STD_LOGIC_VECTOR( 9 DOWNTO 0 );
OUTLed,SevSeg: OUT STD_LOGIC_VECTOR(31 downto 0);
Bottons : in STD_LOGIC_VECTOR(31 downto 0);
Sign_extend          : OUT  STD_LOGIC_VECTOR( 31 DOWNTO 0 );
clock, reset         : IN   STD_LOGIC;
write_register_address_0,write_register_address_1,write_register_address_2
       : out STD_LOGIC_VECTOR( 4 DOWNTO 0 );
write_register_address            : in STD_LOGIC_VECTOR( 4 DOWNTO 0 )
);
```

מסלול קריטי של הרכיב:



יחידת Control unit:

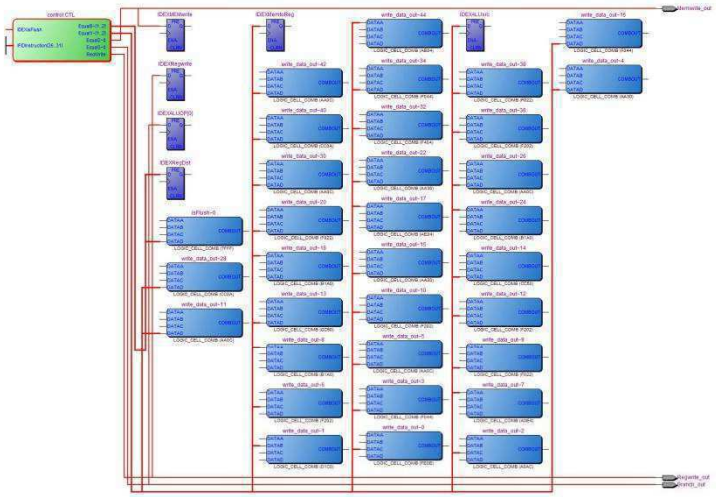
תפקידו של רכיב זה אחראי לפענוח הפקודה לאותות בקרה. כלומר, רכיב זה הינו זה שאחראי ליצור התאמה בין הפקודה שהובאה מהזיכרון לבין אותות הבקרה שיועברו לשלבים וליחידות השונות.כך שלמעשה יודעים השלבים והיחידות מה לעשות עם המידע שהגיע לידיהם.

איור 6

Port map:

```
PORT(  Opcode                          : IN   STD_LOGIC_VECTOR( 5 DOWNTO 0 );
RegDst                          : OUT  STD_LOGIC;
ALUSrc                          : OUT  STD_LOGIC;
MemtoReg                   : OUT  STD_LOGIC;
RegWrite                   : OUT  STD_LOGIC;
MemRead                    : OUT  STD_LOGIC;
MemWrite                   : OUT  STD_LOGIC;
Branch                          : OUT  STD_LOGIC;
ALUop                      : OUT  STD_LOGIC_VECTOR( 1 DOWNTO 0 );
clock, reset          : IN   STD_LOGIC );
```

מסלול קריטי של הרכיב:



איור 7

שלב ה Execute:

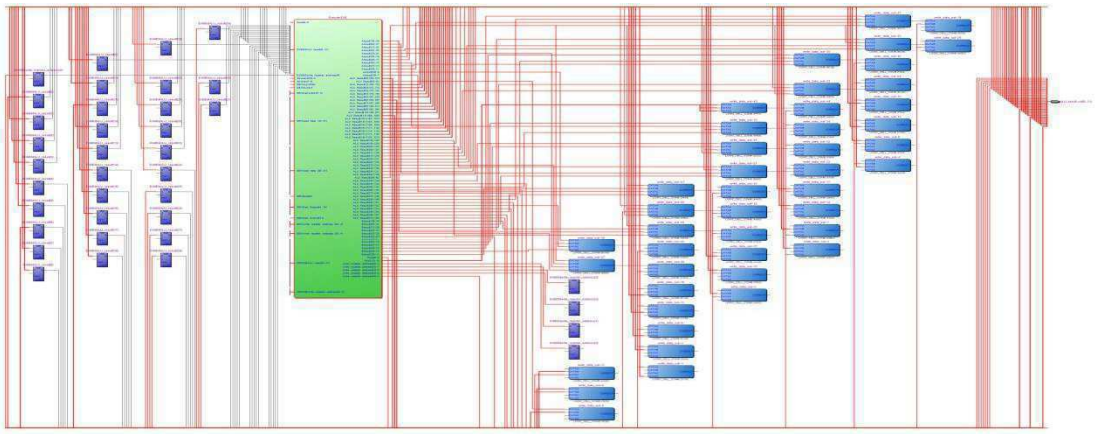
שלב זה הוא השלב השלישי מתוך ארבעת השלבים של המעבד.
תפקיד שלב זה הוא לבצע את הפעולות האריתמטיות והלוגיות בין ערכי הרגיסטרים לבין עצמם, ובין ערכי רגיסטרים לערכים קבועים.
שלב זה מכיל למעשה את רכיב ה ALU כמו זה שעליו עבדנו במעבדה הקודמת.

איור 8

Port map:

```
PORT(  Read_data_1          : IN   STD_LOGIC_VECTOR( 31 DOWNTO 0 );
Read_data_2          : IN   STD_LOGIC_VECTOR( 31 DOWNTO 0 );
Sign_Extend          : IN   STD_LOGIC_VECTOR( 31 DOWNTO 0 );
Function_opcode             : IN   STD_LOGIC_VECTOR( 5 DOWNTO 0 );
ALUOp                       : IN   STD_LOGIC_VECTOR( 1 DOWNTO 0 );
ALUSrc                             : IN   STD_LOGIC;
--Zero                             : OUT  STD_LOGIC;
ALU_Result                  : OUT  STD_LOGIC_VECTOR( 31 DOWNTO 0 );
ALU_ResultFor,ALU_ResultForFor          : IN   STD_LOGIC_VECTOR( 31 DOWNTO 0 );
clock, reset         : IN   STD_LOGIC ;
write_register_address_0,write_register_address_1           : in
STD_LOGIC_VECTOR( 4 DOWNTO 0 );
RegDst                             : IN   STD_LOGIC;
forward,forward2                          : IN STD_LOGIC_VECTOR( 1 DOWNTO 0 );
write_register_address             : out STD_LOGIC_VECTOR( 4 DOWNTO 0 )
```

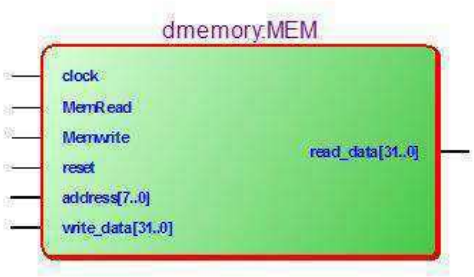
מסלול קריטי של רכיב זה:



איור 9

שלב Memory Write Back :

שלב זה הינו השלב הרביעי של המעבד.
בשלב זה מתבצעת הכתיבה לזיכרון (או הקריאה מהזיכרון), והפניית המילה חזרה לשלב השני, לכתיבה אל הרגיסטרים.

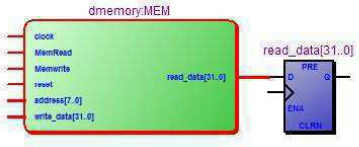


איור 10

Port map:

```
PORT(  read_data                    : OUT  STD_LOGIC_VECTOR( 31 DOWNTO 0 );
address                     : IN   STD_LOGIC_VECTOR( 7 DOWNTO 0 );
write_data                  : IN   STD_LOGIC_VECTOR( 31 DOWNTO 0 );
MemRead, Memwrite    : IN   STD_LOGIC;
Clock,reset                 : IN   STD_LOGIC );
```

מסלול קריטי של רכיב זה:



איור 11

סיכונים כתוצאה ממעבר המעבד לתצורת pipline ודרכי התמודדות:

כתוצאה משינוי יחידת העיבוד המרכזית לתצורת Pipelined MIPS זמן העבודה של המערכת אמנם התקצרה והמערכת הפכה יעילה יותר אך נוצרו מספר בעיות איתם יש להתמודד.
הסיכונים מתחלקים ל 3 סוגים עיקריים אותם למדנו בהרצאה:

- Data Hazards
- Structural Hazards
- Control Hazards

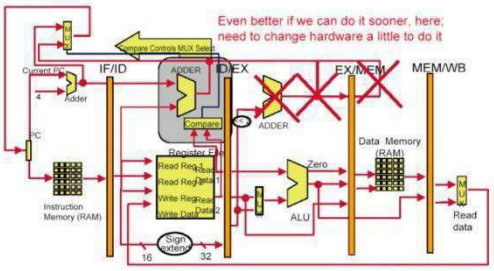
Structural Hazards

סוג סיכונים אלו נוצרים כתוצאה ממספר פניות לאותה חומרה בו זמנית משני מקורות שונים. באופן עקרוני, סיכון זה יכול להתקיים בעבודה עם הזיכרון, כאשר מצד אחד שלב ה IF דורש הבאת פקודה מהזיכרון, ובמקביל שלב ה M&WB דורש הבאת מילה מכתובת בזיכרון. במערכת שלנו לא קיים סיכון מסוג זה, לכן לא היה צורך להתמודד.

Control Hazards

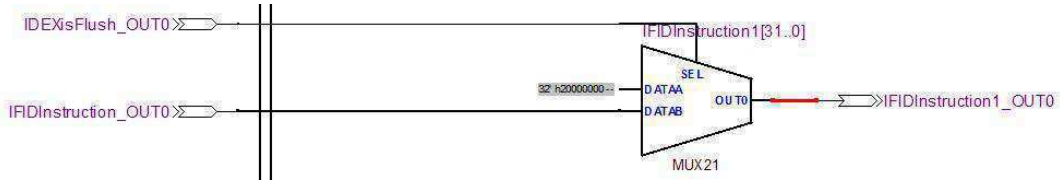
במצב בו קיים בקוד הסתעפות, יכולה להווצר בעיה מכיוון שעד תהליך גילוי הסתעפות עוברות מספר פקודות נוספות וצינור, כאשר פקודות אלו לא בהכרח אמורות להתבצע לפי הלוגיקה של הקוד,

לפתרון בעיה זו נקטנו בשני צעדים, צעד ראשון הוא לפענח האם צריכה ההסתעפות להתבצע "מוקדם יותר", כך:



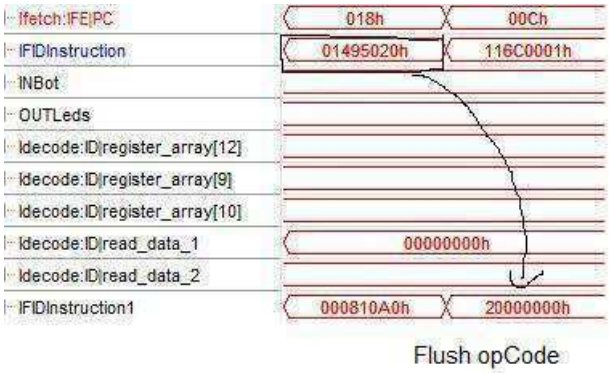
איור 12

כך שלמעשה אנו מפענחים הסתעפות כבר בשלב פענוח הפקודה עצמה, לכן תכנס לכל היותר פקודה אחת אשר לא צריכה להתבצע בוודאות להתבצע, כדי לפתור בעייה זו, השתמשנו בגישת flush, כלומר הפיכת פקודה זו לריקה-nop לפני ביצוע. עשינו זאת על ידי קו בקרה אשר קובע האם בוצעה הסתעפות.



איור 13

Single tap של בעיה זו:



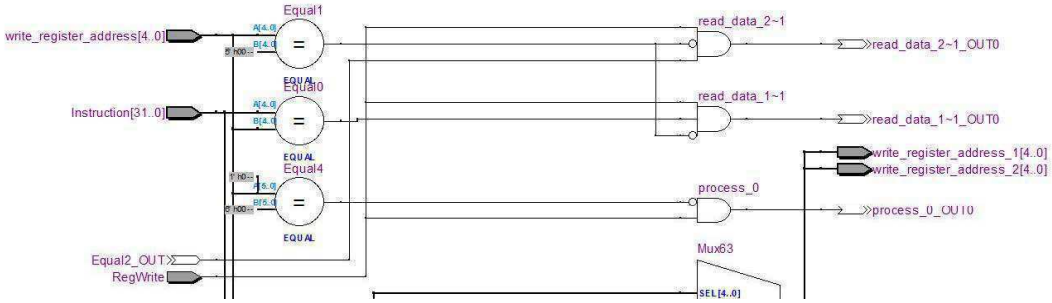
איור 13

3.1.3 סיכוני מידע (Data Hazards)

סיכוני המידע נוצרים כאשר ישנו שימוש בערך רגיסטר אשר תוצאתו עדיין לא התעדכנה מפקודות קודמות שבוצעו עליו.

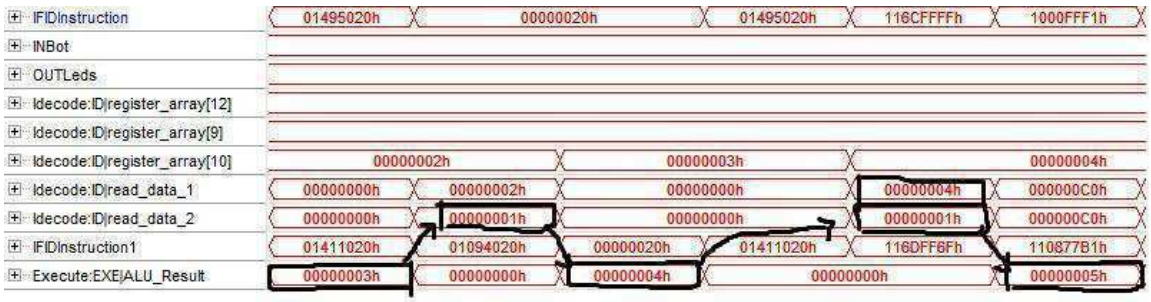
בעיה ראשונה: התנגשות בקריאה וכתיבה לרגיסטר, בעיה זו קורה כאשר נכנסת לצינור פקודה הטוענת ערך לרגיסטר מסויים, ולאחר 2 פקודות כלשהם, נכנסת פקודה המבקשת להשתמש בערך השמור ברגיסטר, נוצר מצב של צורך לקרוא ולכתוב לרגיסטר באותו מחזור שעון.

כדי להתמודד עם בעיה זו יצרנו יחידה המזהה את המצב ובמקור לכתוב ולקרוא, אנו מעבירים ישירות את מה שאנו רוצים לכתוב לקריאה.



איור 14

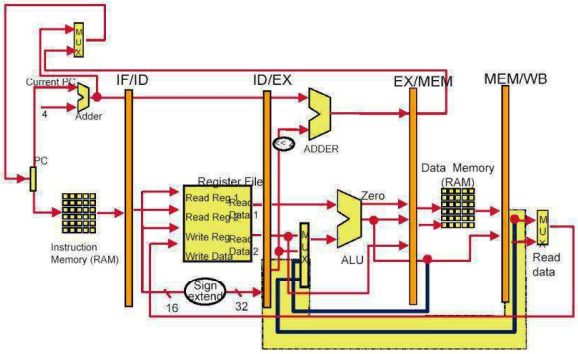
Single tap לבעיה זו:



בעיה שנייה: המקרה הראשון הוא כאשר מתבצעות 2 פקודות ברצף, כאשר התוצאה של הפקודה הראשונה היא המידע של הפקודה הבאה אחריה. במקרה כזה הפתרון יתבצע ע"י הוספת יחידת קידום (Forwarding) אשר תחזיר את התוצאה ממוצא ה ALU, ישירות לשלב הקודם.

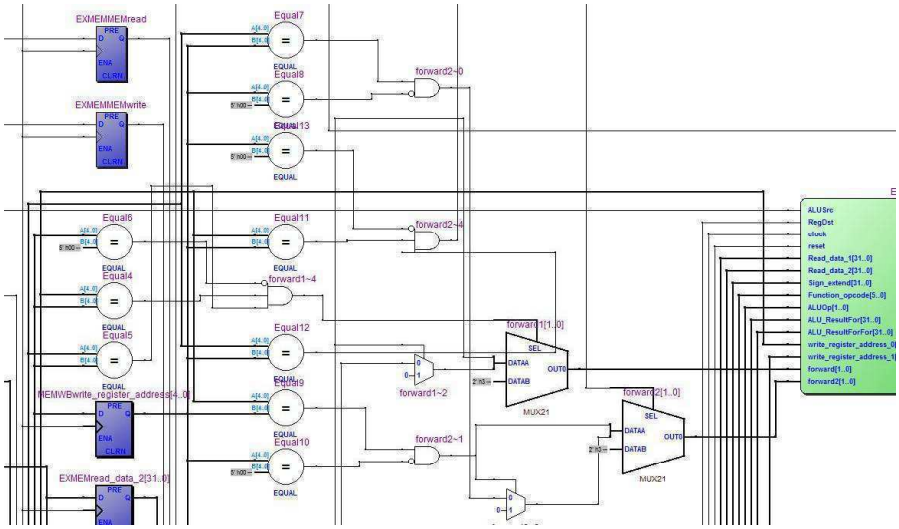
בנוסף בעיה דומה היא כאשר שני הפקודות לעיל כאשר בינהם פקודה כלשהי החוצצת בינהם, ההבדל בטיפול בשני הבעיות הוא במספר מחזורי השעון שצריך לעכב את התוצאה מהALU,

כפי שתואר בהרצאה :



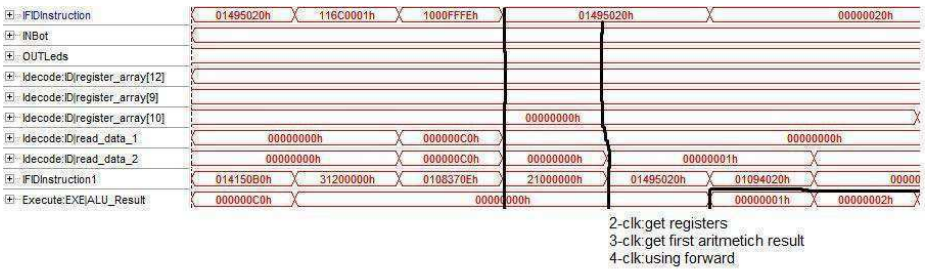
איור 15

במימוש שלנו:



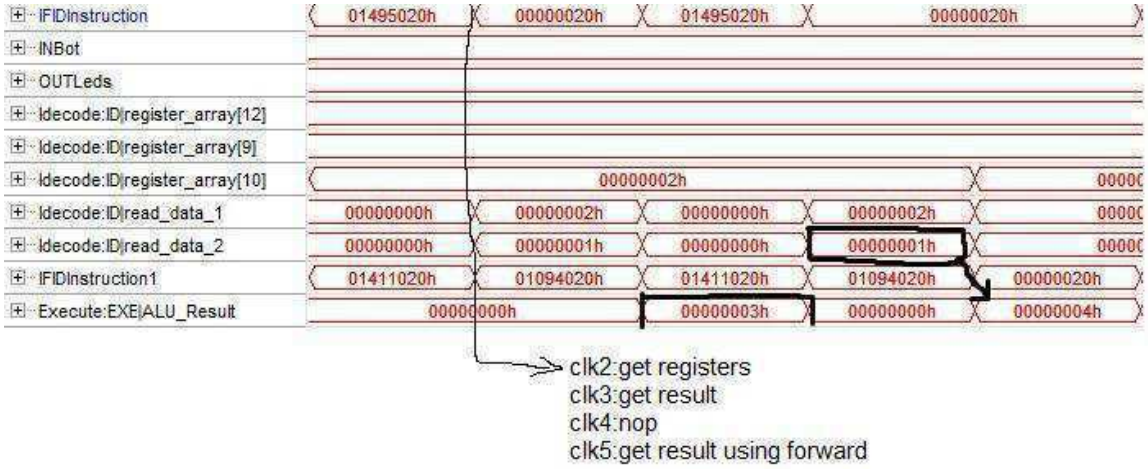
איור 16

Single tap לטיפול בשתי פקודות ברצף :



איור 17

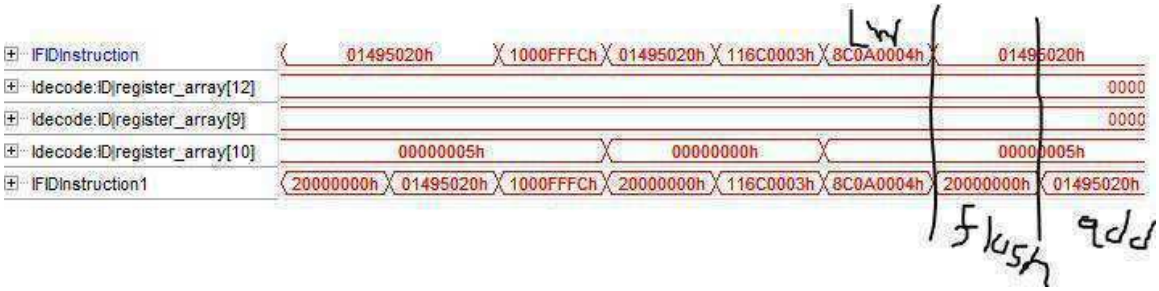
Single tap לטיפול בשתי פקודות עם פקודה חוצצת בינהם :



איור 18

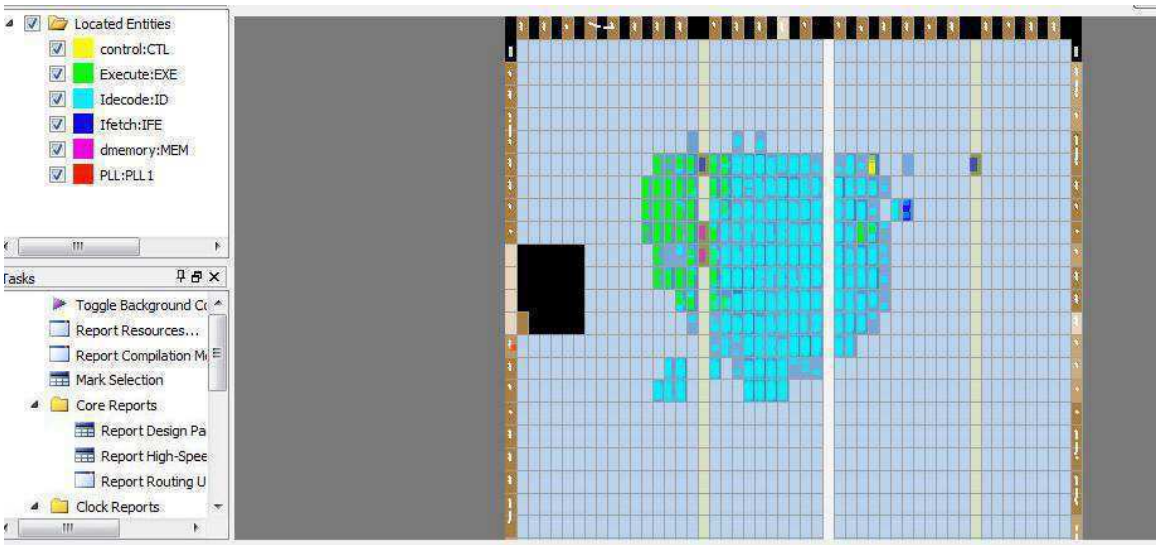
בעיה אחרונה שיכולה להווצר היא כאשר אנו מבצעים טעינה לתוך רגיסטר ומיד לאחר מכן מנסים להשתמש בו, כדי לפתור בעיה זו יצרנו יחידה הבודקת האם מתקיים מצב כזה, במקרה וכן אנו מעכבים את שעון המערכת מלהתקדם ודוחפים "בועה", פקודה ריקה כדי לעקב את המערכת ולא ליצור בעיה.

Single tap של בעיה זו :



איור 19

Chip planner (logic usage):



איור 20

Flow Status	Successful - Tue May 28 18:26:39 2013
Quartus II 64-Bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	MIPS
Top-level Entity Name	MIPS
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	2,213 / 18,752 (12 %)
Total combinational functions	2,149 / 18,752 (11 %)
Dedicated logic registers	1,312 / 18,752 (7 %)
Total registers	1312
Total pins	272 / 315 (86 %)
Total virtual pins	0
Total memory bits	16,384 / 239,616 (7 %)
Embedded Multiplier 9-bit elements	0 / 52 (0 %)
Total PLLs	1 / 4 (25 %)

איור 21

שעונים ותדר מערכת מקסימאלי:

Clocks

	Clock Name	Type	Period	Frequency	Rise	Fall	Duty Cycle	Divide by	Multiply by
1	altera_reserved_tck	Base	100.000	10.0 MHz	0.000	50.000			
2	clock	Base	20.000	50.0 MHz	0.000	10.000			
3	PLL 1|altpll_component|pll|clk[0]	Generated	30.769	32.5 MHz	0.000	15.384	50.00	20	13

איור 22

Slow Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	37.87 MHz	37.87 MHz	PLL 1|altpll_component|pll|clk[0]	
2	84.84 MHz	84.84 MHz	clock	

איור 23

מסקנות

העבודה הייתה קשה מאוד מבחינת היקף ורמת העבודה .

קשה מאוד היה גם לעבוד על העבודה בזוג בו זמנית.

למרות זאת העבודה שילבה בתוכה ידע רב בתכנות, במעבד ה MIPS, בתוכנות השונות, בפייפליין ועוד.

עצם מורכבות העבודה היה אתגר בפני עצמו.

נוכחנו במהלך העבודה לעוצמה הרבה של תכנון ותכנות שבבי FPGA ונפתחנו לעולם של תכנון ה CPU.

אייר ודודו