CPU Architecture

Final Project Report File

עומר לוקסמבורג 205500390

עילי נוריאל 312538580

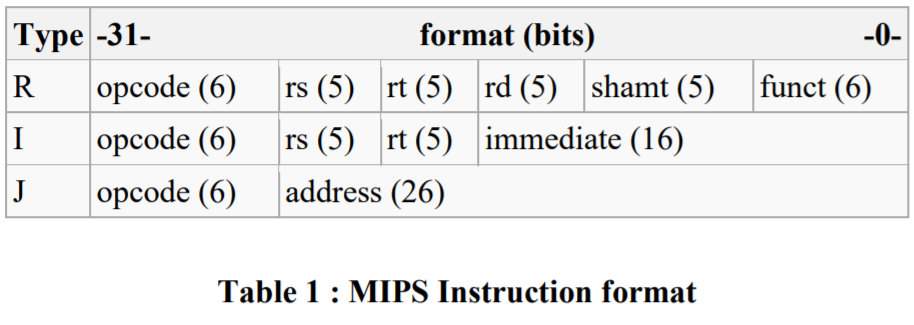
# **מטרת המטלה**

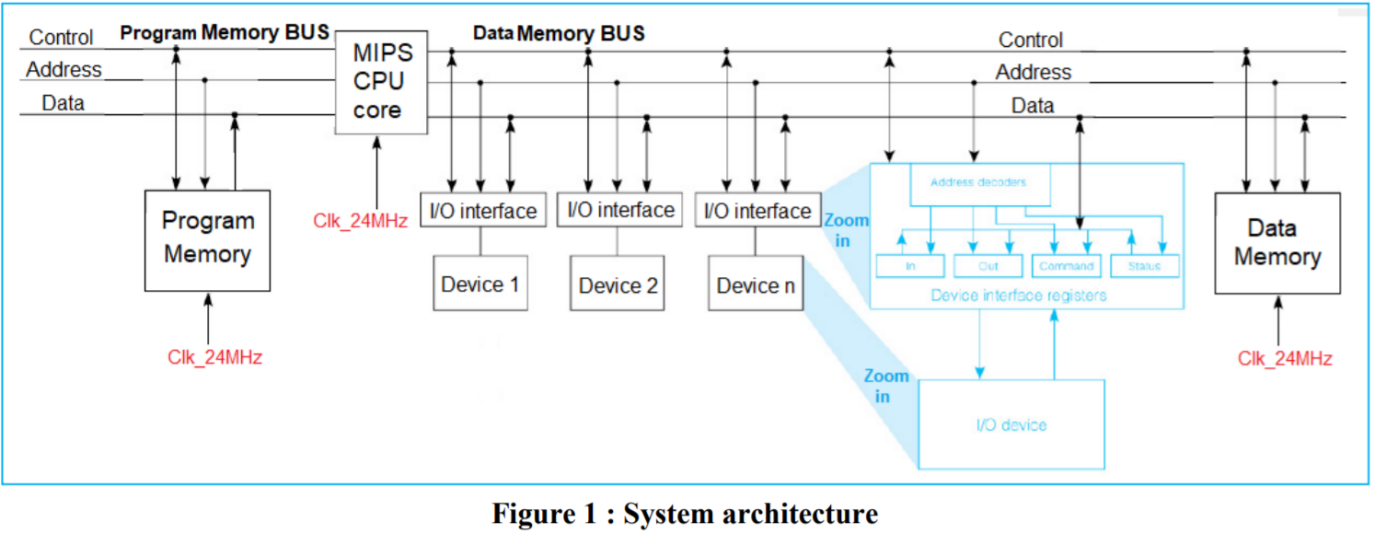
מטרת מטלה זו היא ביצוע תכנון, סינתזה וניתוח של CPU פשוט מסוג MIPS, עם שילוב של Mapped I/O, וכמו כן הבנת מבנה הזיכרון של רכיב ה- Cyclone II FPGA. כמו כן הוספת עבודה עם טיימר ובקר פסיקות.

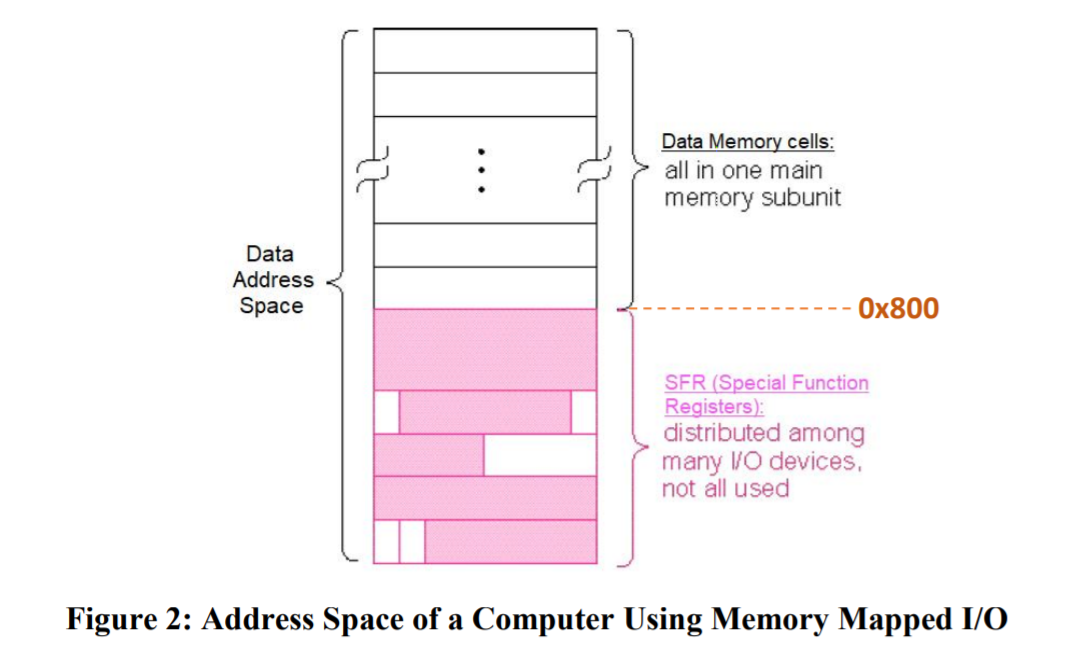
# **הגדרת תכנון המערכת**

ה-CPU אותו תכננו עובד בתצורת Single Cycle, ומבצע פקודות מה-ISA של ה-MIPS.

כמו כן ל-CPU יש register file סטנדרטי של MIPS, וה- top level entity מתוכנן בתצורת structural.

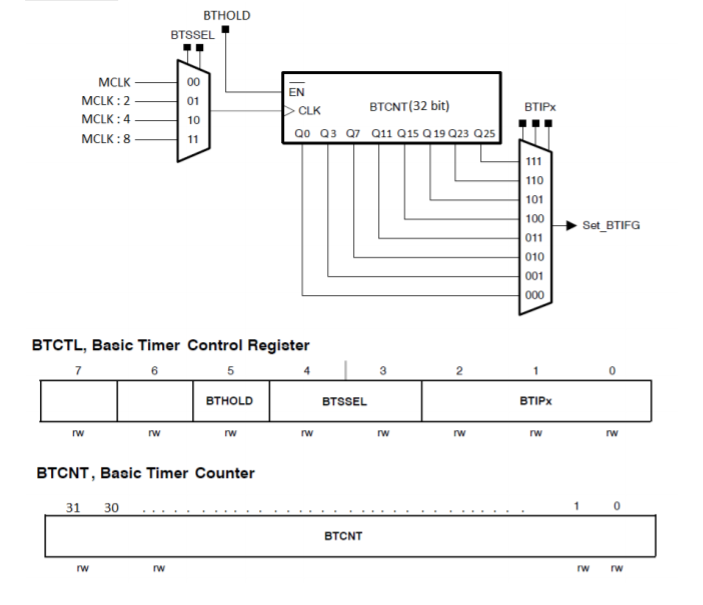
פקודות נוספות (מעבר למטלה 3) שמימשנו הן: JR, JAL והן תוכננו בהתאם לפורמט הפקודות המתואר בTable 1-:

התכנון אותו ממשנו תואם לדיאגרמת הבלוקים המתוארת ב-Figure 1:

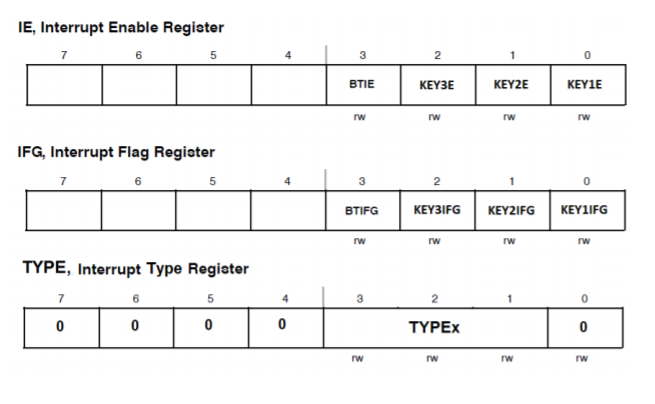
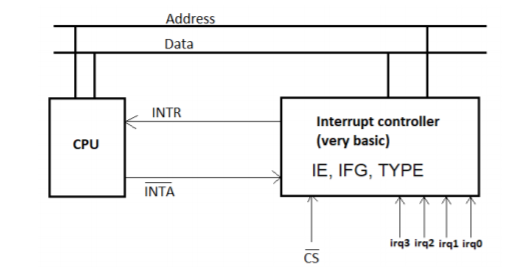
כמו כן התכנון שלנו מותאם ל-Mapped I/O המקושר למרחב הכתובות המתואר ב- Figure 2:

# **פריפריות נדרשות**

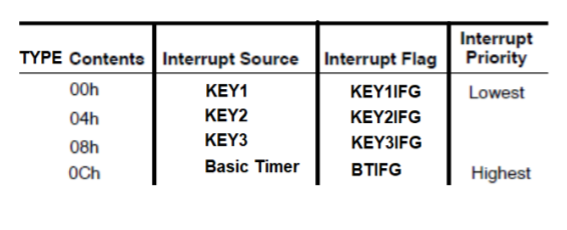
בנוסף לפריפריות שנדרשו מאיתנו במטלה 3 (נורות הלד, המתגים ומסכי הספרות) נדרשנו להוסיף:

1. כפתורים 1-3.
2. Basic Timer - טיימר בסיסי בעל שני רגיסטרים (רגיסטר Control ורגיסטר Counter) שתוכנן בהתאם ל-Figure 3:

**Figure 3: Basic Timer Structure**

1. בקר פסיקות – המקבל בקשות לפסיקה מ-4 מקורות (כפתורים והטיימר) ובעל 3 רגיסטרים (רגיסטר Enable, רגיסטר דגלים, ורגיסטר Type לסוג הפסיקה) שתוכנן בהתאם ל-Figure 4:

**Figure 4: Interrupt Controller Structure**

כמו כן תיעדוף בין הפסיקות השונות יתבצע לפי Table 2:

**Table 2: Interrupt Controller Priority Table**

**הערות:**

* נדרש מאיתנו לתכנן בקר פסיקות ללא קינון (Nesting), על כן הוספנו גם חומרה המוודאת כי לא מתרחש קינון ומגינה על המערכת במקרה של בקשות פסיקה המתרחשות בו זמנית.

# **אופן מימוש החומרה הנוספת**

1. מימוש הפקודות JR, JAL –
   1. **JR** – בדומה למימוש פקודת JUMP, אנו משתמשים בקווי בקרה ייעודיים (isJR) היוצאים ממודול ה-control ל-IEXECUTE. כאשר מזוהה כי מתבצעת פקודת JR, ניקח את ADD\_Result (תוצאה של ALU המחשב Branchים, במקרה שלנו זה ערך כתובת ב-words אליה נקפוץ) הנלקח מרגיסטר הייעד שמצוין ב- Instruction, ונקפוץ אל הכתובת השמורה בו ע"י השמה ל-PC.
   2. **JAL** – על מנת לבצע פקודה זו, אנו שומרים את כתובת ה-PC אליה אנו נרצה לחזור (PC+4) ברגיסטר מספר 31 ($ra). במקביל אנו מבצעים את פקודת JUMP, כלומר קופצים לערך הכתובת המיידית הנתונה ב-Instruction ב-words.
2. מימוש ה-BasicTimer –

על מנת לממש את החלוקות הנדרשות מהמודול, הגדרנו מונה באורך 3 ביטים, כך שכל שעון מחולק יהיה תלוי בערך הביט הרלוונטי לו במונה. למשל MCLK2 מקבל את ערכו לפי הביט במיקום ה-0 במונה (כאשר הוא עולה ל-1 גם השעון עולה ל-1 וכו').

הביט BTIFG – יוצא מהמודול ומתחבר ל-IRQ3 בבקר הפסיקות, עולה ל-1 כאשר נוצר overflow בקבוצת הביטים שנבחרה (לפי BTIPx). לדוגמא אם נבחר ביט Q3, BTIFG יעלה ל-1 כאשר נקבל overflow עבור ביטים Q0-Q3, כלומר כאשר ספרנו  *פעמים.*

1. מימוש ה-InterruptController – השרשרת המלאה מתוארת בתרשים הזרימה הכחול מטה.
   1. ראשית, הגדרנו למודול קו ENABLE "המאזין" ל-Address Bus, Data Bus ומאפשר את המודול כאשר פונים לכתובת שלו.
   2. הגדרנו ביט GIE (הביט ברגיסטר $k0 במקום ה-0). כאשר ביט זה שווה ל-0, המודול למעשה מושבת ולא יטופלו פסיקות.
   3. ניתן לרשום ולקרוא מהרגיסטרים IE, IFG לפי הכתובות שהוגדרו.

# **בדיקת התכנון והמערכת**

ניתוח עבודת המערכת מתבצע ע"י שני ה- test benchים הבאים, שניתנו לנו מראש:

1. תמיכה והרצה תקינה של הקוד בקובץ ה-*test1.asm* שניתן לנו – בדיקת מערך הקריאה לפונקציה, פקודות jal ו-jr:

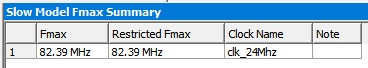
חיבור 2 מספרים השמורים בזכרון, X,Y, ושולחים את הסכום לפונקציה המחסרת ממנו 4 ומחזירה את הערך. במידה והערך חיובי התוכנית מחזירה אותו, במידה והוא שלילי – היא מחזירה את הסכום המקורי ללא חיסור 4.

1. תמיכה והרצה תקינה של הקוד בקובץ ה-*test2.asm* שניתן לנו – קריאת הערך של המתגים והצגתו באופנים שונים בהתאם למקור הפסיקה:
   * Key 1 – מציג את הערך של ה-SW + הערך שהמונה ספר על גבי הLEDים (הירוקים והאדומים).
   * Key 2 – מציג את ערך הערך של ה- SW + הערך שהמונה ספר על גבי Hex 0,1.
   * Key 3– מציג את ערך הערך של ה- SW + הערך שהמונה ספר על גבי Hex 0,1.
   * BTIFG – הוספת 1 לערך המונה והצגת הערך על גבי הLEDים הירוקים.

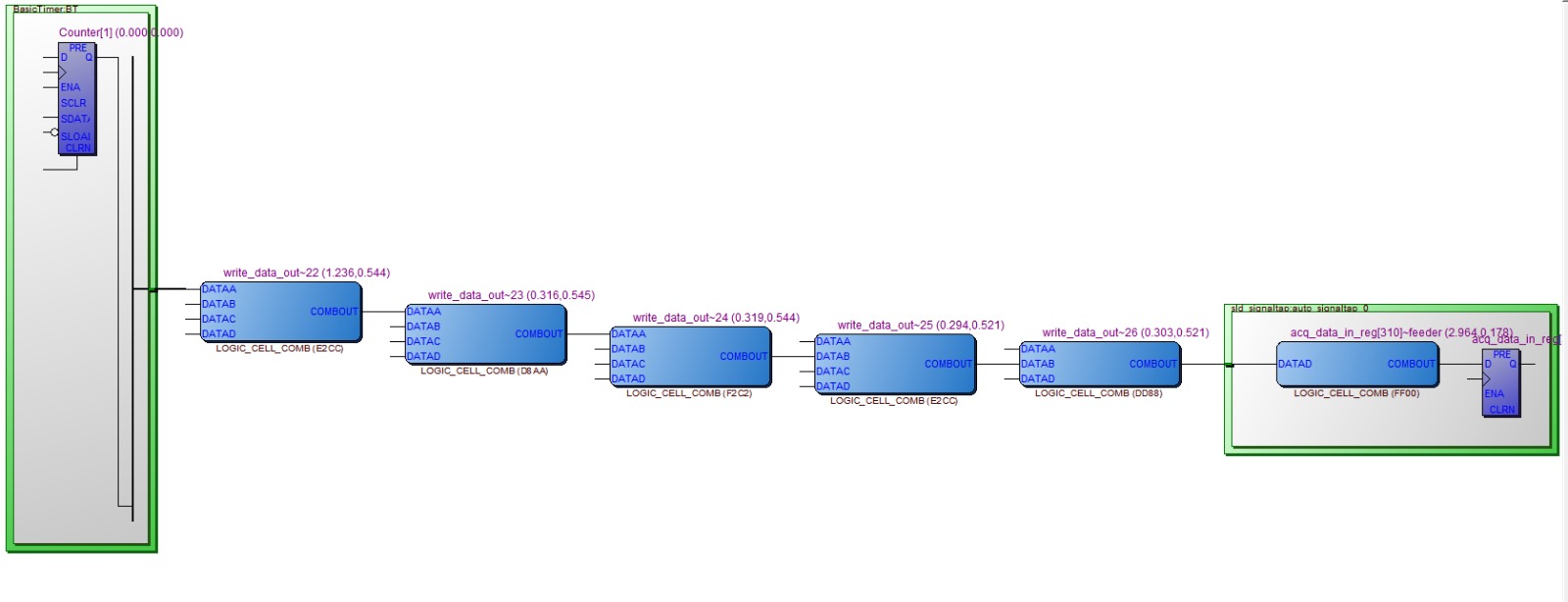
**הערות:**

* הוספנו מנגנון המאפשר מעבר בין תוכנת ה-ModelSim לתוכנת ה-Quartus ללא צורך בשינוי הקוד, כך שה-tester שב-ModelSim מזריק ערך המצביע על כך שאנו בModelSim. זאת ע"י מימוש גנרי למודולים הדורשים זאת (MIPS, IFETCH).

# **ניתוח עבודת המערכת**

* **תדר השעון המקסימלי** שקיבלנו הוא :

**Figure 5: Fmax analysis**

* **ניתוח המסלול הקריטי:**

**Figure 6: Critical Path**

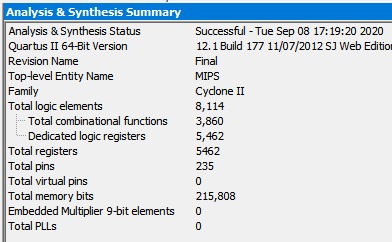
המסלול הקריטי כולל שימוש ב-Counter – שמצריך שמירה של ערכו על מנת לחשב את המצב הבא, וגם חיבור של 2 מספרים באורך של 32 ביטים.

* **ניתוח המסלול המינימלי:**

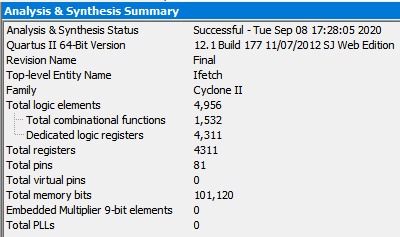
אנו משערים כי המסלול הכי קצר המערכת הוא של אחת מפעולות ה-bitwise, OR או AND (XOR ככל הנראה מעט ארוכה יותר), זאת מכיוון שהן מבוצעות ע"י מימוש פשוט של שער לוגי וללא פעולות אריתמטיות מסובכות, וכמו כן הן נכתבות לרגיסטר ולא לזיכרון.

באופן כללי – כלל הפעולות מבצעות את 4 השלבים של המערכת (כאשר יש פעולות שמבצעות Write back לרגיסטרים ויש פעולות הכותבות לזיכרון) כלומר, אין פעולה קצרה משמעותית מפני שכולן עוברות בשלבי המערכת כפי שלמדנו על מעבד single cycle.

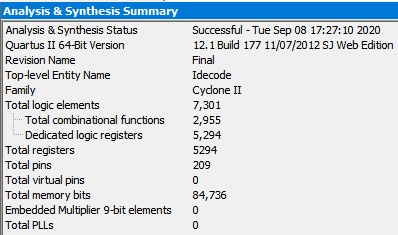
* **ניתוח השימוש הלוגי:**

של ה-MIPS כולו:

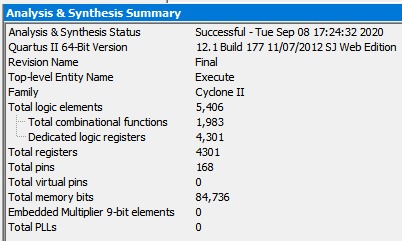
**Figure 7: MIPS Logic Usage**

****Fetch:

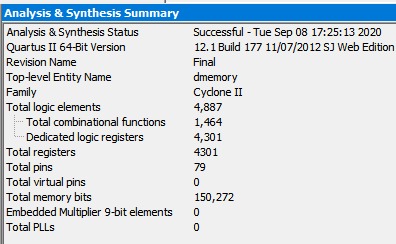
**Figure 8: Ifetch Logic Usage**

****Decode:

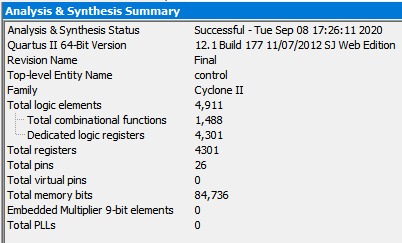
**Figure 9: Idecode Logic Usage**

Execute:

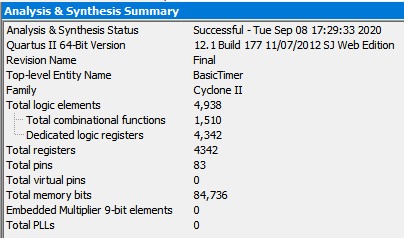
**Figure 10: Execute Logic Usage**

Memory:

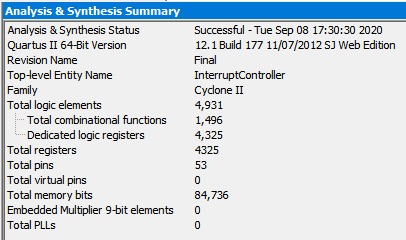
**Figure 11: dmemory Logic Usage**

:Control

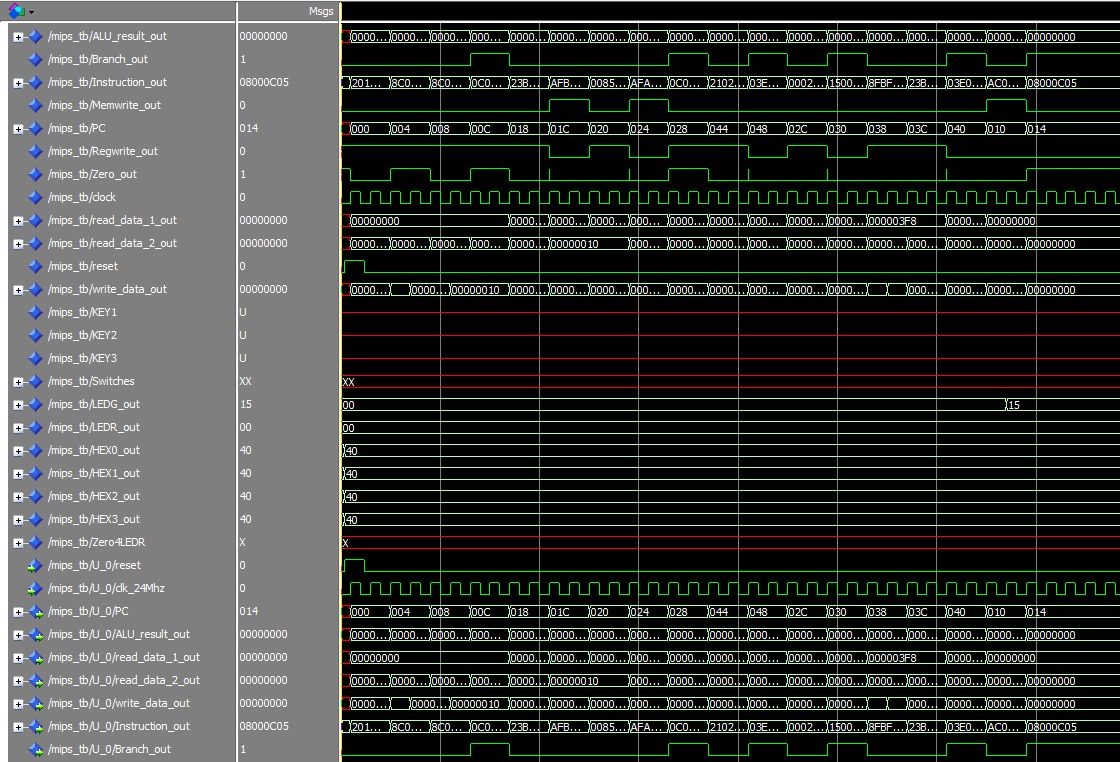
**Figure 12: control Logic Usage**

:BasicTimer

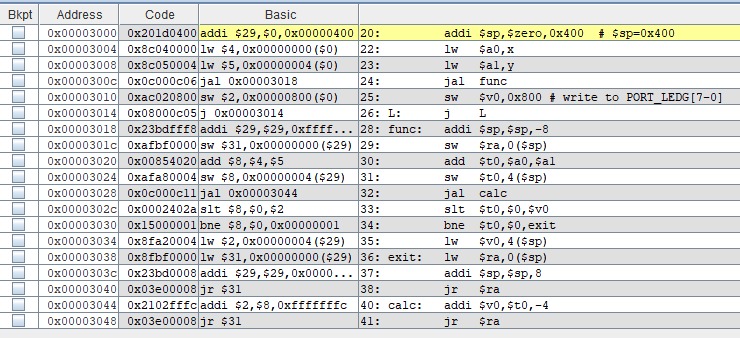
**Figure 13: BasicTimer Logic Usage**

:InterruptController

**Figure 14: InterruptController Logic Usage**

* **Wave form:** זוהי תצוגה של הקוד של test1.asm**:**

**Figures 15: test1 Wave form analysis**

****

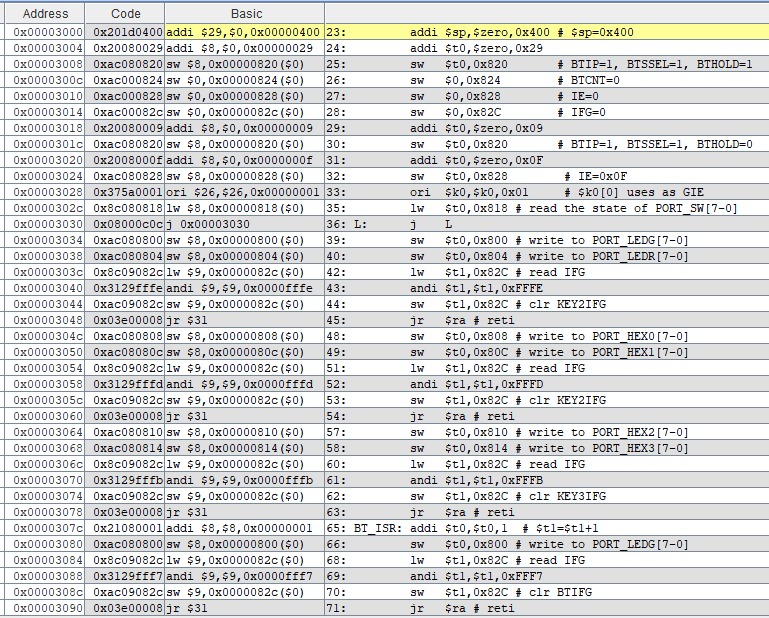
בשורה 0x3040 מתבצעת פקודת JR לכתובת 0x3010, ואכן ב-Waveform ניתן לראות כי ערך ה-PC קופץ לערך הנדרש

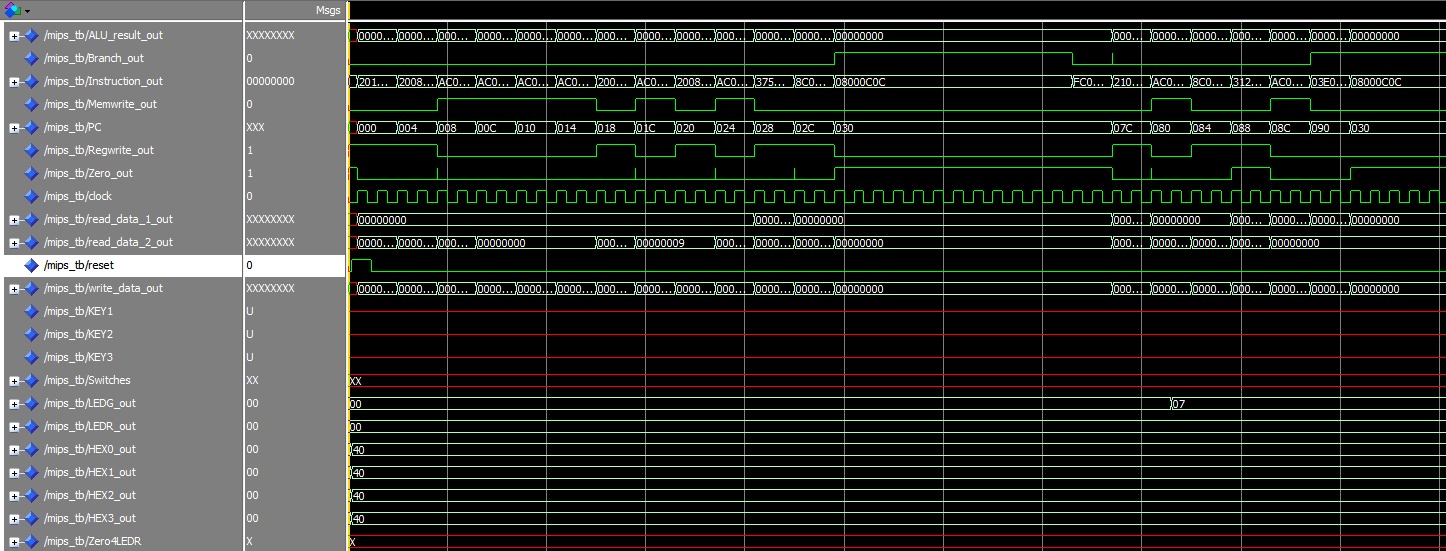
בשורה 0x300C מתבצעת פקודת JAL לכתובת 0x3018, ואכן ב-Waveform ניתן לראות כי ערך ה-PC קופץ לערך הנדרש

**Figures 16: test1 Assembly Code**

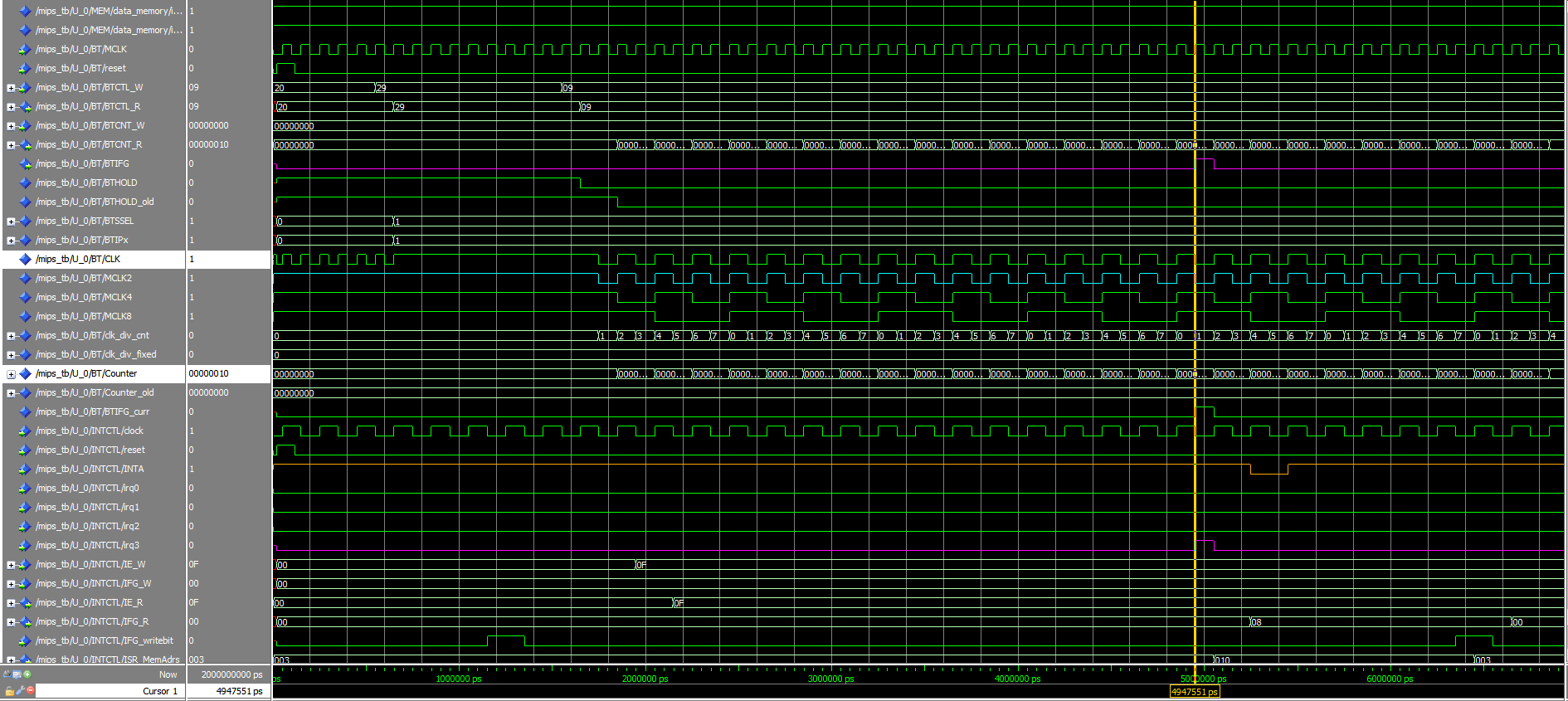
* **Wave form:** זוהי תצוגה של הקוד של test2.asm**:**

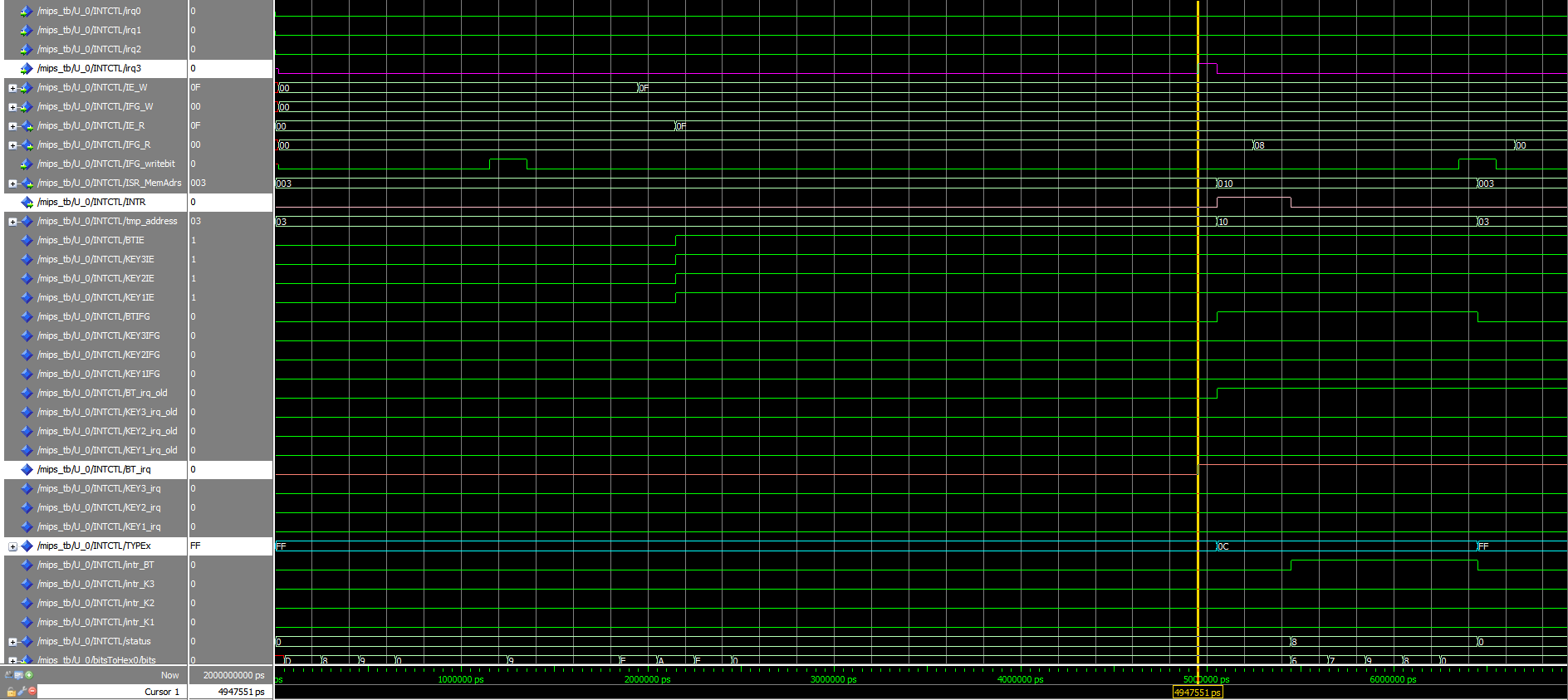
**הערות:**

* + ערכנו את קוד האסמבלי של test2 לשם הבדיקה – הגדרנו BTIP=1, BTSSEL=1 על מנת שהפסיקות מהטיימר יהיו מהירות יותר (חלוקה קטנה יותר של השעון).
  + ****המקרה המתואר ב- 18Figure בוחן רק את תפקוד פסיקת הטיימר מאחר וכל הפסיקות זהות ורצינו לבדוק את עבודת שתי הפריפריות החדשות יחדיו, תוך ניצול כלל חלקי המערכת.

**Figures 17: test2 Assembly Code**

ניתן לראות כי כל עוד אין פסיקה, התוכנית נשארת על פקודה מספר 0x3030. כאשר מתרחשת פסיקה של הטיימר, אנו מקבלים מה-InterruptController חיווי – ובעקבותיו מתחילה שרשרת הפעולה לטיפול בפסיקה. כחלק משרשרת זו הגדרנו את Opcode מספר 31, שאינו נמצא בשימוש אחר, כ-Opcode יעודי לפסיקות. ניתן לראות כי אכן הפקודה שמתבצעת היא בעלת Opcode 0xFC.





בתכלת – השעון שבחרנו.

בסגול העליון – דגל הפסיקה של הטיימר.

ניתן לראות כי BTIFG עולה ל-1 כאשר ערך ה-Counter מגיע ל-16 (*.(*

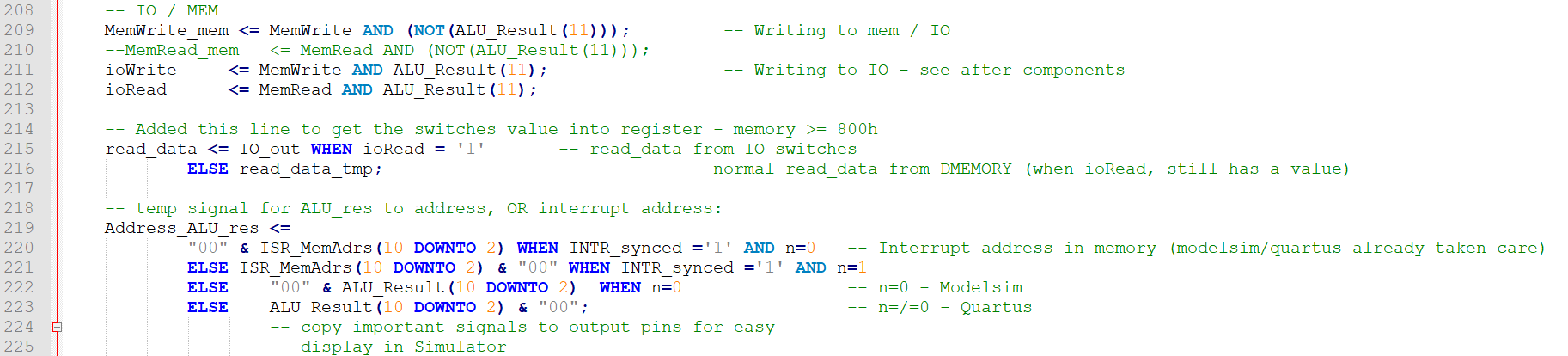
*בכתום – INTA – יורד ל-0 מחזור אחד לאחר שקיבלנו פסיקה.*

**Figures 18-20: test2 Wave form analysis**

בסגול– בקשת הפסיקה של הטיימר IRQ3.

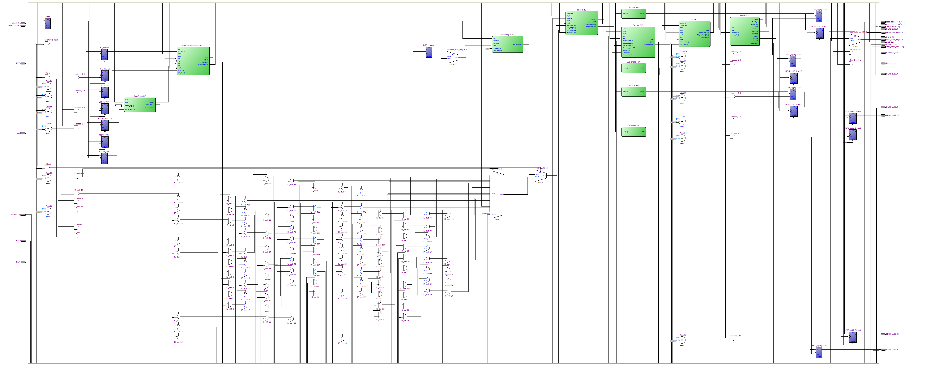
ניתן לראות כי intr\_BT עולה ל-1 (מתרחש כאשר נכנסים את הפסיקה הרלוונטית) והוא מונע מפסיקות נוספות להתבצע*.*

* **RTL viewer:**

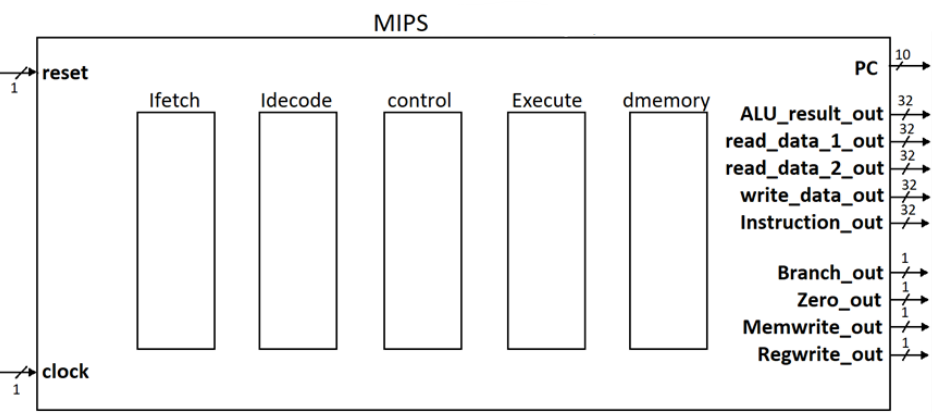
ניתן להבחין ביחידות לוגיות המטפלות בניתוב הקריאה והכתיבה בין כתובות זיכרון רגילות לבין כתובות זיכרון הממופות ל- I\O:

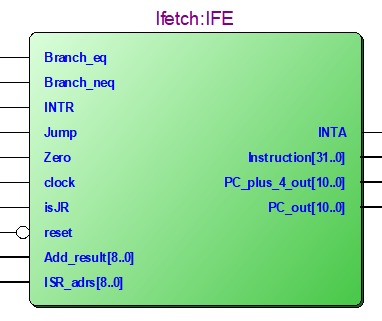
**Figure 21: IO/MEM Mapping**

ניתן לראות כי הוספנו קווי בקרה בעזרתם אנו יודעים האם פונים ל- I\O.

**דיאגרמת ה-RTL המלאה של המערכת [](https://drive.google.com/file/d/12SHrVk_PEWWZegsVEeBih0n9ai07veTe/view?usp=sharing)**

**Figure 22: RTL view**

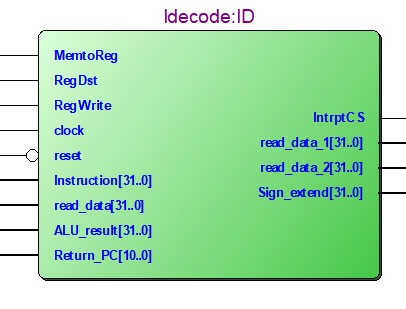
להלן הבלוקים המתוארים בדיאגרמת ה-RTL בפירוט גדול יותר – כולל כניסות ויציאות, בהתאם לדיאגרמה המתארת את המערכת ללא הפריפריות:

**Figure 23: Top Block Diagram**

Ifetch

שליפת הפקודה הבאה מהזיכרון בהתאם לקווי הבקרה הנכנסים והוצאת הפקודה הבאה.

**Figure 24: Ifetch block**

****

**Figure 25: Idecode block**

Idecode

קבלת הפקודה הגולמית מה-Ifetch והוצאת סיגנלי מידע שאותם נצטרך בשביל ביצוע הפקודה.

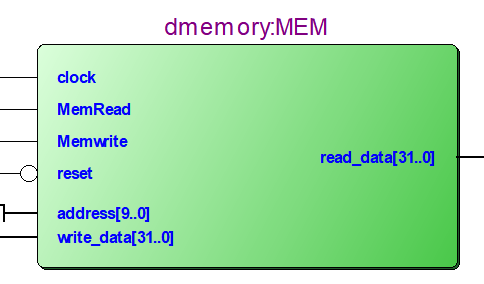
****

**Figure 26: Execute block**

Execute

ביצוע הפקודה עצמה, כלומר פעולה אריתמטית (מתוך אחת מהפעולות הנתמכות במעבד). והוצאת תוצאה ודגלים מתאימים.

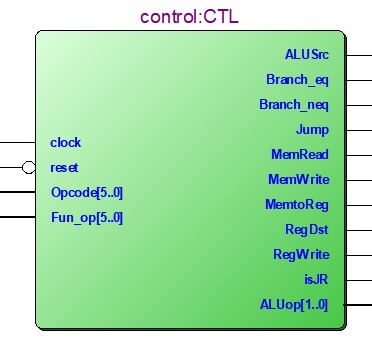
Add\_Result – תוצאה המייצגת כתובת ולכן הוא רק באורך של שמונה ביטים.

****

dmemory

מבצע קריאה או כתיבה לזכרון, מלבד לכתובות הממופות לרכיבי ה-I\O

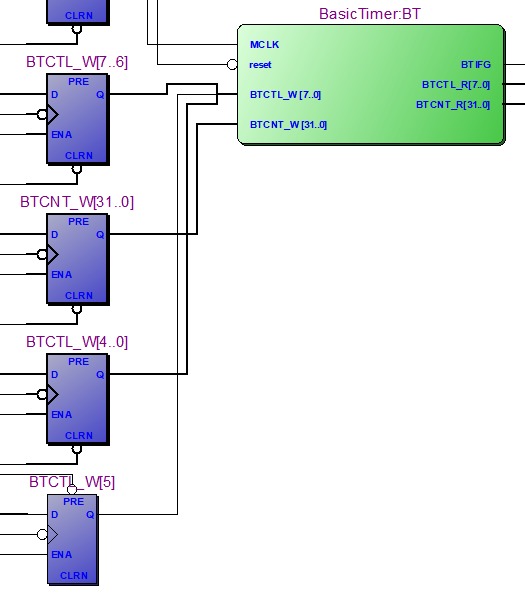
**Figure 27: dmemory block**

****

control

הוצאת קווי הבקרה המתאימים לפקודה (לפי ה-opcode המגיע מה(fetch-

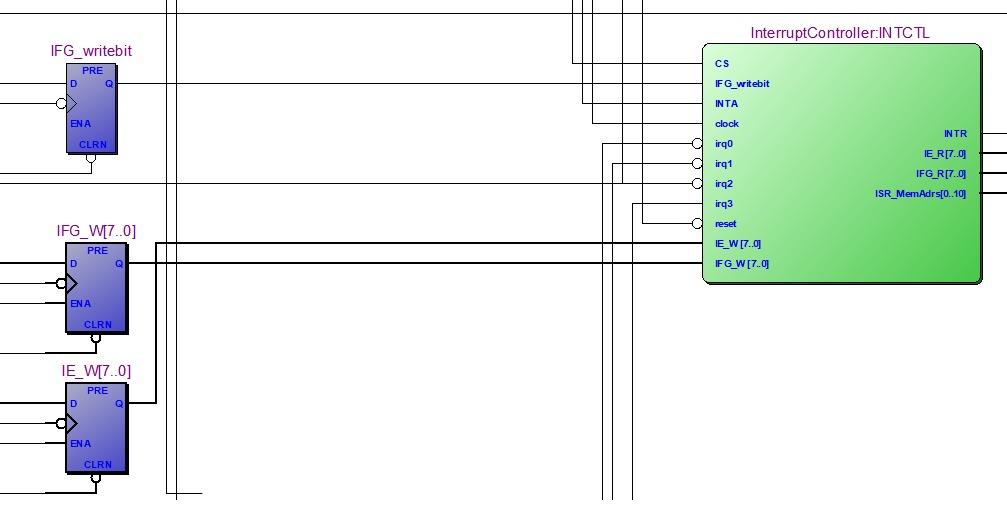
**Figure 28: control block**

****

BasicTimer

טיימר - רכיב פריפריאלי – הניתן להגדרה עם חלוקות שונות של השעון – ובעל counter פנימי. בעל יכולת פסיקה

**Figure 29: BasicTimer block**

****

InterruptController

בקר פסיקות - רכיב פריפריאלי – בעל יכולת קבלה של פסיקות מ-4 מקורות שונים.

**Figure 30: InterruptController block**

**Port Table:** מצורפות טבלאות המתארות את הגודל, כיוון ופונקציונאליות של הקווים היוצאים מכל בלוק במערכת

**Table 3: MIPS** **ports**

|  |  |  |  |
| --- | --- | --- | --- |
| IFETCH | | | |
| Port | Direction | Size | Functionality |
| Instruction | out | 32 |  |
| PC\_plus\_4\_out | out | 11 | The next command address |
| Add\_result | in | 9 | Address value to branch / jump to |
| Branch\_eq | in | 1 | Control signal indicates BEQ command |
| Branch\_neq | in | 1 | Control signal indicates BNQ command |
| Zero | in | 1 |  |
| Jump | in | 1 | Control signal indicates Jump command |
| isJR | In | 1 | Control signal indicates Jump Register command |
| PC\_out | out | 11 |  |
| INTR | In | 1 |  |
| INTA | Out | 1 |  |
| ISR\_adrs | In | 9 |  |
| clock | in | 1 |  |
| reset | in | 1 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| MIPS | | | |
| Port | Direction | Size | Functionality |
| reset | in | 1 |  |
| clk\_24Mhz | in | 1 |  |
| PC | out | 11 |  |
| ALU\_result\_out | out | 32 |  |
| read\_data\_1\_out | out | 32 |  |
| read\_data\_2\_out | out | 32 |  |
| write\_data\_out | out | 32 |  |
| Instruction\_out | out | 32 |  |
| Branch\_out | out | 1 |  |
| Zero\_out | out | 1 |  |
| Memwrite\_out | out | 1 |  |
| Regwrite\_out | out | 1 |  |
| ENABLE | In | 1 |  |
| KEY1 | In | 1 |  |
| KEY2 | In | 1 |  |
| KEY3 | In | 1 |  |
| Switches | in | 8 |  |
| LEDG\_out | buffer | 8 |  |
| LEDR\_out | buffer | 8 |  |
| HEX0\_out | buffer | 7 |  |
| HEX1\_out | buffer | 7 |  |
| HEX2\_out | buffer | 7 |  |
| HEX3\_out | buffer | 7 |  |
| HEX4\_out | buffer | 7 |  |
| Zero4LEDR | out | 2 | To initiate LEDR 8 and 9 to 0 value. |

**Table 4: Ifetch** **ports**

|  |  |  |  |
| --- | --- | --- | --- |
| IDECODE | | | |
| Port | Direction | Size | Functionality |
| read\_data\_1 | out | 32 | Data #1 to use in execute |
| read\_data\_2 | out | 32 | Data #2 to use in execute |
| Instruction | in | 32 |  |
| read\_data | in | 32 |  |
| ALU\_result | in | 32 |  |
| RegWrite | in | 1 |  |
| MemtoReg | in | 1 |  |
| Reg\_dst | in | 1 |  |
| Sign\_extend | out | 32 | Sign extended immediate |
| IntrptCS | Out | 1 |  |
| ReturnPC | In | 11 | ערך ה-PC שאליו נחזור לאחר ביצוע פסיקה |
| clock | in | 1 |  |
| reset | in | 1 |  |

**Table 5: Idecode** **ports**

|  |  |  |  |
| --- | --- | --- | --- |
| IExecute | | | |
| Port | Direction | Size | Functionality |
| Read\_data\_1 | in | 32 | Data #1 to read from |
| Read\_data\_2 | in | 32 | Data #2 to read from |
| Sign\_extend | in | 32 | Sign extended immediate |
| Function\_opcode | in | 6 | For “0” opcode |
| Opcode | in | 6 | Opcode number |
| ALUOp | in | 2 | With Function\_opcode, I\_opcode – decides which ALU operation will be executed |
| Jump | in | 1 |  |
| isJR | in | 1 |  |
| ALUSrc | in | 1 |  |
| Zero | out | 1 |  |
| ALU\_Result | out | 32 | The result of the ALU (shift included) |
| Add\_Result | out | 9 |  |
| PC\_plus\_4 | in | 11 |  |
| Clock | in | 1 |  |
| Reset | in | 1 |  |

**Table 6: IExecute** **ports**

|  |  |  |  |
| --- | --- | --- | --- |
| DMEMORY | | | |
| Port | Direction | Size | Functionality |
| read\_data | out | 32 | Memory data read |
| address | in | 11 | Address number in |
| write\_data | in | 32 | Data to write |
| MemRead | in | 1 | ONLY for memory read (no I/O) |
| Memwrite | in | 1 | ONLY for memory write (no I/O) |
| clock | in | 1 |  |
| reset | in | 1 |  |

**Table 7: IDmemory** **ports**

|  |  |  |  |
| --- | --- | --- | --- |
| CONTROL | | | |
| Port | Direction | Size | Functionality |
| Opcode | in | 6 |  |
| Fun\_op | In | 6 |  |
| RegDst | out | 1 | What register to write to |
| ALUSrc | out | 1 | Binput selector |
| MemtoReg | out | 1 |  |
| RegWrite | out | 1 |  |
| MemRead | out | 1 | Read control |
| MemWrite | out | 1 | Write control |
| Branch\_eq | out | 1 |  |
| Branch\_neq | out | 1 |  |
| ALUop | out | 2 |  |
| Jump | out | 1 |  |
| isJR | Out | 1 |  |
| clock | in | 1 |  |
| reset | in | 1 |  |

**Table 8: Control** **ports**

|  |  |  |  |
| --- | --- | --- | --- |
| BasicTimer | | | |
| Port | Direction | Size | Functionality |
| MCLK | In | 1 |  |
| reset | In | 1 |  |
| BTCTL\_W | in | 1 | Basic timer control write to |
| BTCTL\_R | Out | 1 | Read from |
| BTCNT\_W | in | 1 | Basic timer counter write to |
| BTCNT\_R | Out | 1 | Read from |
| BTIFG | Out | 1 | Out flag  **Table 9: BasicTimer ports** |

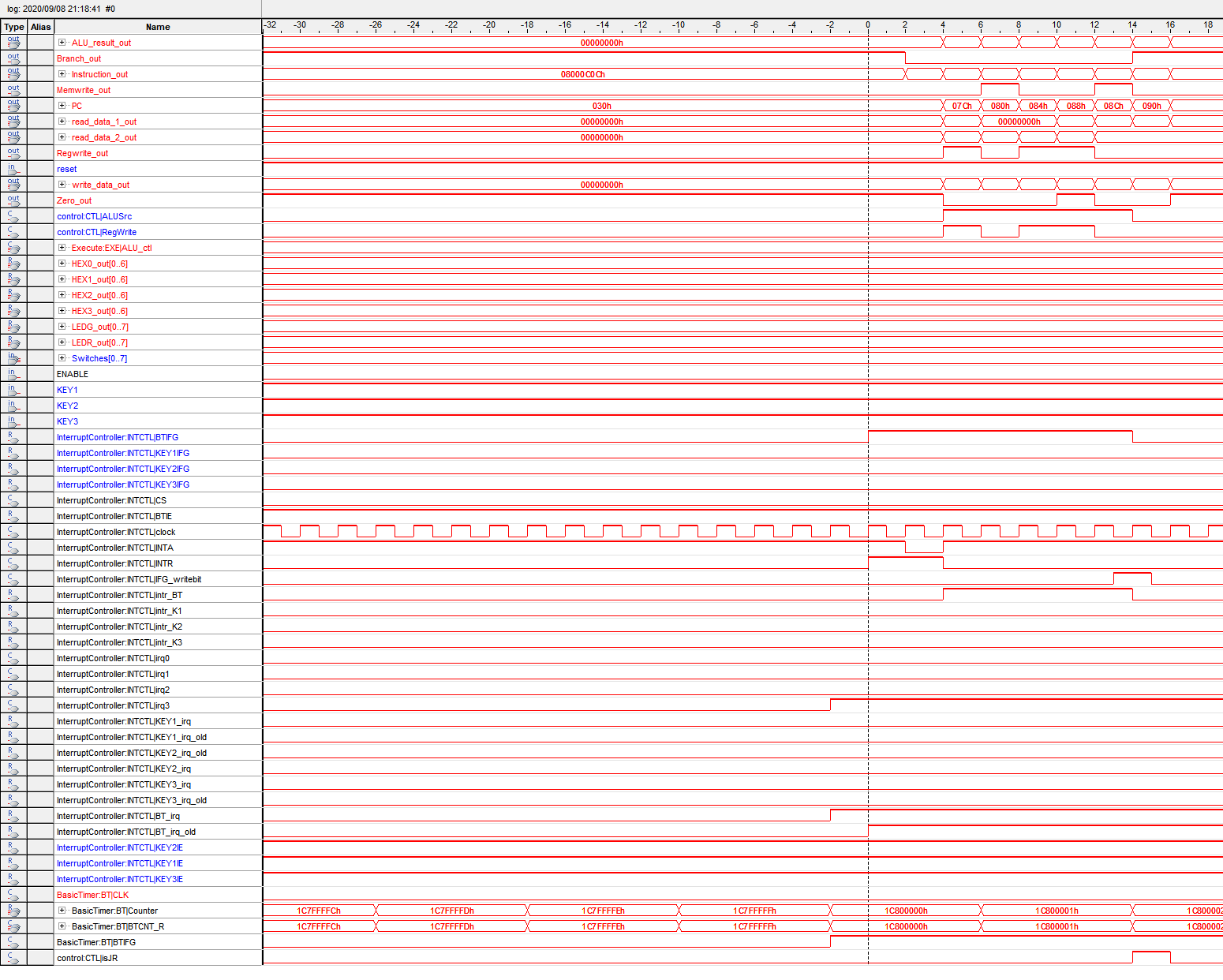
|  |  |  |  |
| --- | --- | --- | --- |
| InterruptController | | | |
| Port | Direction | Size | Functionality |
| clock | In | 1 |  |
| reset | In | 1 |  |
| INTA | In | 1 | Interrupt Acknowledge |
| irq0 | In | 1 |  |
| irq1 | In | 1 |  |
| irq2 | In | 1 |  |
| irq3 | In | 1 |  |
| IE\_W | In | 8 | Interrupt enable write to |
| IFG\_W | In | 8 | Interrupts flags |
| IE\_R | Out | 8 | Read from |
| IFG\_R | Out | 8 | Read from |
| IFG\_writebit | In | 1 | Notify bit |
| ISR\_MemAdrs | Out | 11 | Address of ISR in dmemory |
| INTR | Out | 1 | Interrupt request |

**Table 10: InterruptController** **ports**

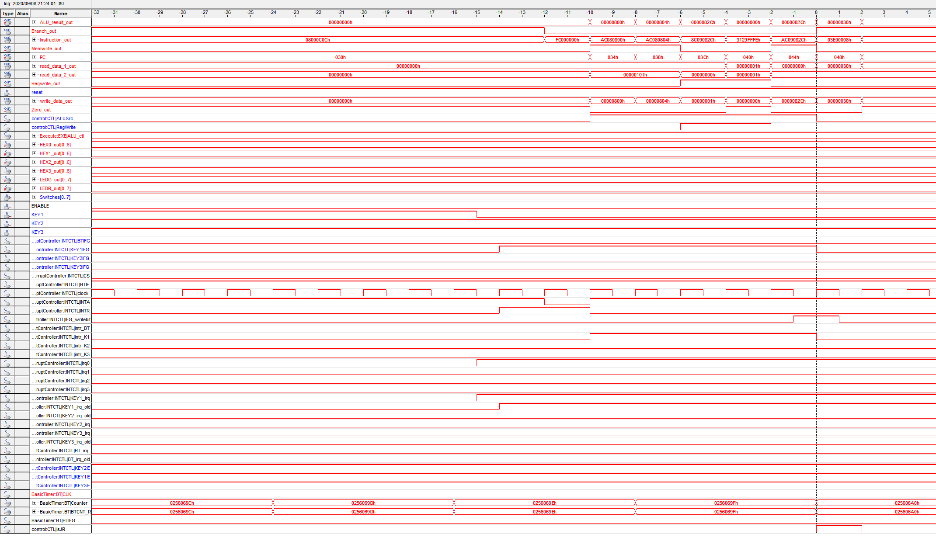
|  |  |  |  |
| --- | --- | --- | --- |
| I/O Memory (ports in MIPS.vhdl file) | | | |
| Port | Direction | Size | Functionality |
| MemWrite\_mem | signal | 1 | ‘1’ when Memwrite=’1’ and ALU\_result is **not** the address of the I/O. |
| ioWrite | signal | 1 | ‘1’ when Memwrite=’1’ and ALU\_result **is** the address of the I/O. |
| ioRead | signal | 1 | ‘1’ when MemRead=’1’ and ALU\_result **is** the address of the I/O. |
| read\_data | signal | 32 | Out will be from switches, when ioRead=’1’, else it will be “ DMEMORY - read\_data” signal. |

**Table 11: I/O memory** **ports**

* **Proof of work – Signal Tap Screenshots:**

של test2.asm – עם טריגר המוגדר עבור BTIFG (פסיקת הטיימר)

ניתן לראות כי הפקודה הזו היא פעולת ה-add בה מכניסים לרגיסטר $9 את הערך 0+0 ולכן גם נקבל Zero\_out עם ערך 1.

:

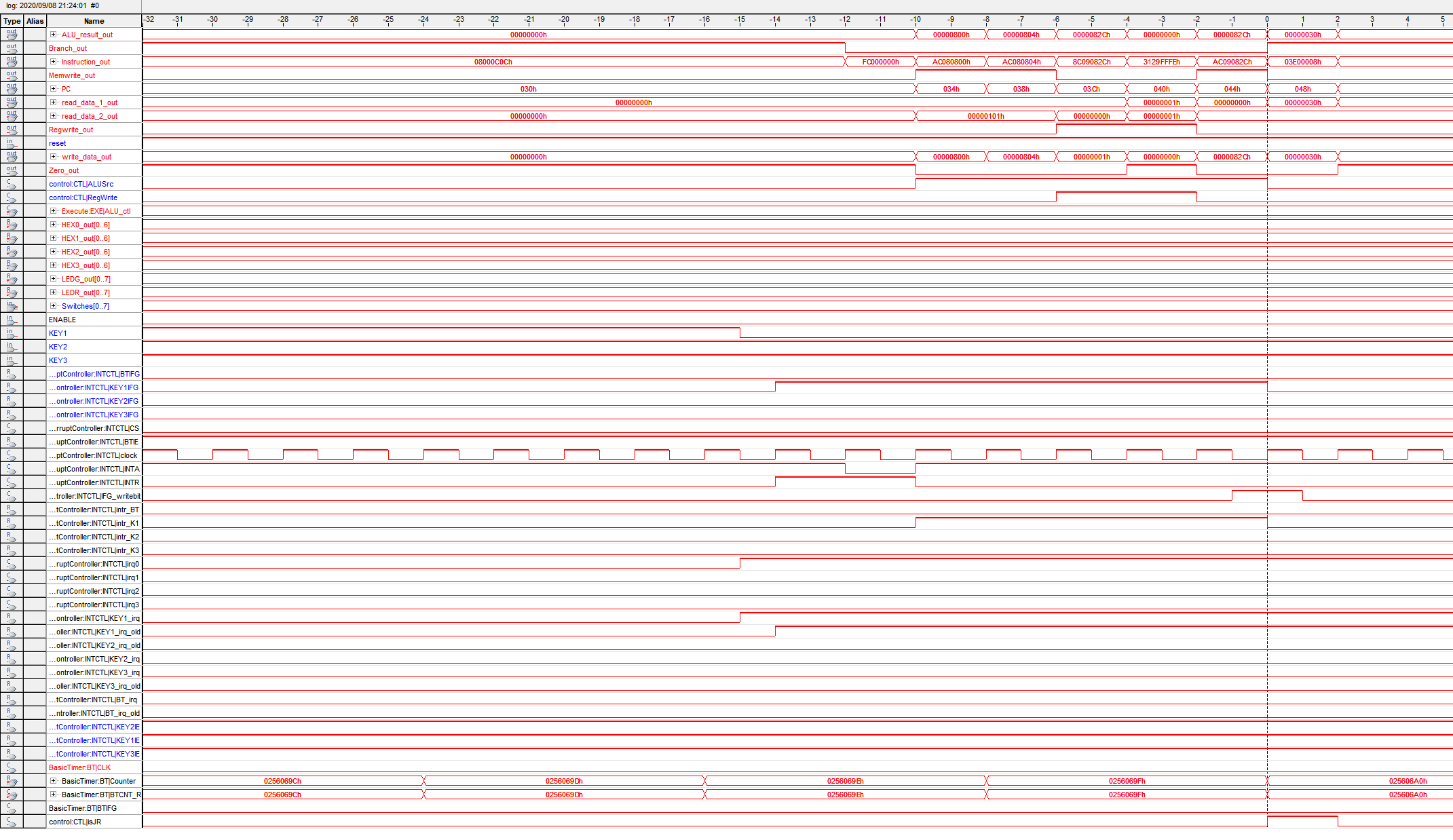
**Figure 26: Assembly Sorting Code**

"מתקבלת" הפקודה המלאכותית לטיפול בפסיקה (OPCODE 31)

ניתן לראות כי בזמן זה יוצאת בקשת פסיקה (INTR) מבקר הפסיקות אל ה-MIPS מכיוון ש- BTIFG בבקר הפסיקות עלה.

ניתן לראות כי בזמן זה עולה ה-BTIFG (היוצא מהטיימר) ולכן גם BT\_irq עולה (בקר הפסיקות מקבל בקשה לפסיקה).

**Figure 31: Signal Tap Basic Timer Interrupt**



ניתן לראות כי בזמן זה התבצעה לחיצה על KEY1 ולכן גם irq0 עולה

**Figure 32: Signal Tap – Key 1 Interrupt**