# PREPARATION REPORT LAB 3

# ADVANCED CPU ARCHITECTURE AND HARDWARE

## Simple RISC Multi-Cycle CPU design modeling

Tal Adoni – 319087300

Omri Aviram – 312192669

במעבדה זו היה עלינו לממש מעבד מולטי-סייקל אשר מכיל שני רכיבים עיקריים.
הרכיב הראשון הינו רכיב ה-Control שמהווה "המוח" של המעבד ולאחר אתחול התוכנית ע"י TestBench מתאים, זיכרון בצורת טקסט (DTCM) וקובץ פקודות (ITCM) נקבל כי בעזרת אותות Control המתקבלים מה-Control אל הרכיב השני, ה-DataPath שבו קיימים הרכיבים המתאימים ובאמצעותם מתבצעת הפונקציונליות הנדרשת, ניתן לבצע תכנית assembly שלמה ע"י שילוב שני הרכיבים האלה בהינתן כי ההוראות הן הוראות שהמעבד מכיר.

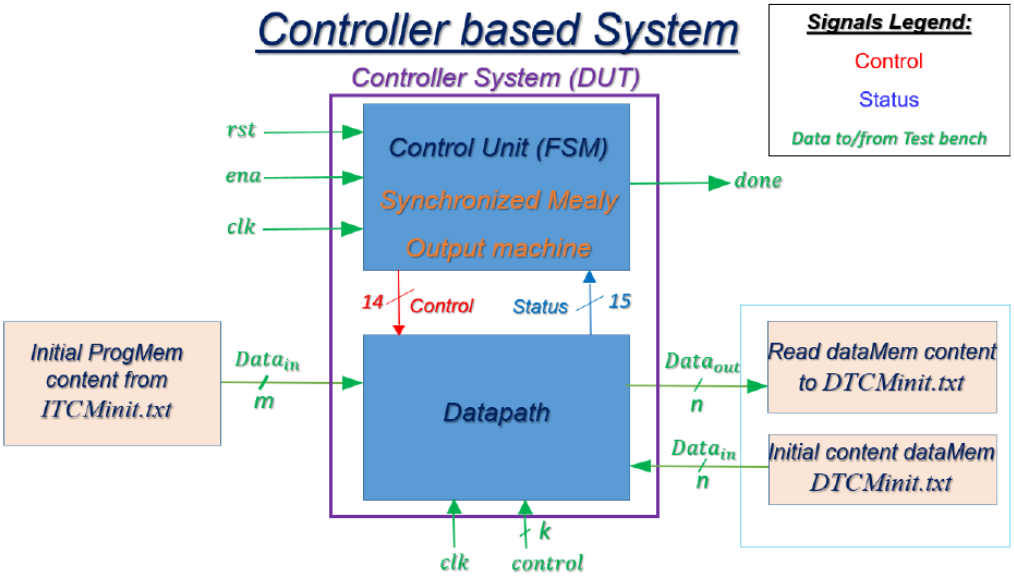



**Figure 1: Overall DUT structure**

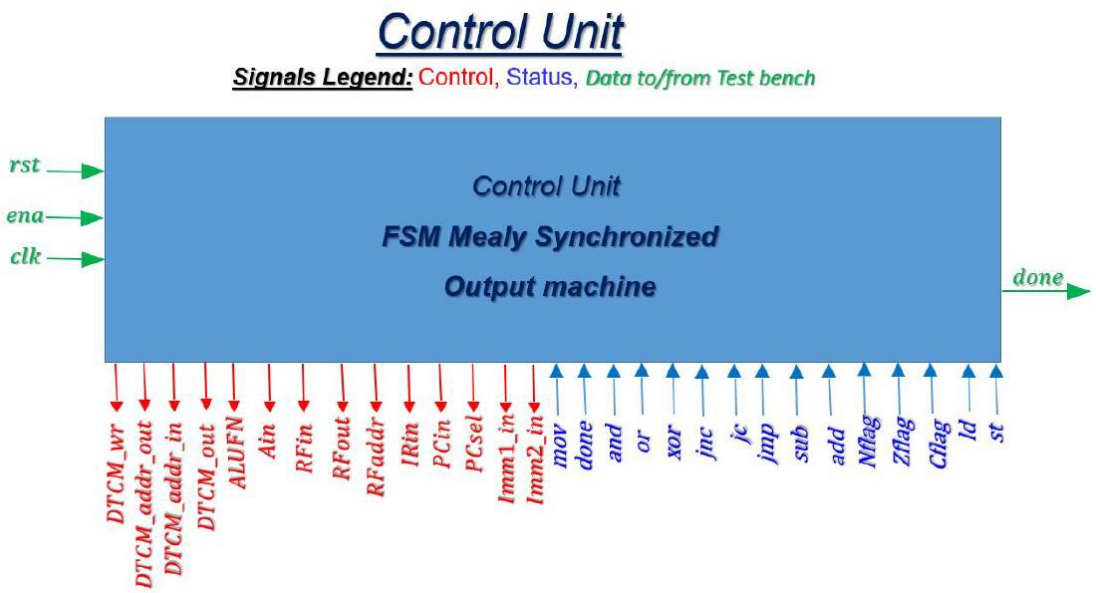
הפקודות אותן היה עלינו לבצע הן:

| Instruction Format | Decimal value | OPC | Instruction | Explanation | N | Z | C |
|---|---|---|---|---|---|---|---|
| R-Type | 0 | 0000 | add ra,rb,rc | R[ra]<=R[rb]+R[rc] | * | * | * |
| | | | nop | R[0]<=R[0]+R[0] (*emulated instruction*) | * | * | * |
| | 1 | 0001 | sub ra,rb,rc | R[ra]<=R[rb]-R[rc] | * | * | * |
| | 2 | 0010 | and ra,rb,rc | R[ra]<=R[rb] and R[rc] | * | * | - |
| | 3 | 0011 | or ra,rb,rc | R[ra]<=R[rb] or R[rc] | * | * | - |
| | 4 | 0100 | xor ra,rb,rc | R[ra]<=R[rb] xor R[rc] | * | * | - |
| | 5 | 0101 | *unused* | | | | |
| | 6 | 0110 | *unused* | | | | |
| J-Type | 7 | 0111 | jmp offset_addr | PC<=PC+1+offset_addr | - | - | - |
| | 8 | 1000 | jc /jhs offset_addr | If(Cflag==1) PC<=PC+1+offset_addr | - | - | - |
| | 9 | 1001 | jnc /jlo offset_addr | If(Cflag==0) PC<=PC+1+offset_addr | - | - | - |
| | 10 | 1010 | *unused* | | | | |
| | 11 | 1011 | *unused* | | | | |
| I-Type | 12 | 1100 | mov ra,imm | R[ra]<=imm | - | - | - |
| | 13 | 1101 | ld ra,imm(rb) | R[ra] <= M[imm+R[rb]] | - | - | - |
| | 14 | 1110 | st ra,imm(rb) | M[imm+R[rb]] <= R[ra] | - | - | - |
| Special | 15 | 1111 | done | Signals the TB to read the DTCM content | - | - | - |

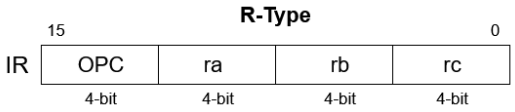
Note: * The status flag bit is affected , - The status flag bit is not affected

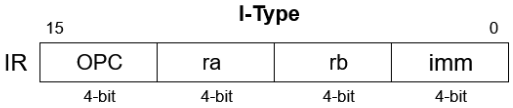
כאשר הפקודות שהינן unused הן פקודות אשר יינתנו לנו במהלך מטלת זמן-אמת.

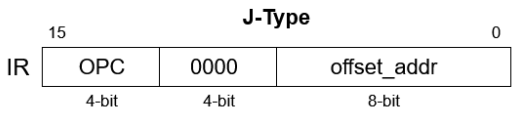
כפי שציינו קודם, יח' Control משמשת כמוח של המערכת והיא זו ששולחת את אותות הבקרה לצורך ביצוע הפעולה. יחידה זו מקבלת מס' אותות כפי שניתן לראות באיור המצורף.
אותות ה-TB הן אותות כניסה של rst, ena, clk אשר בעזרתן אנו קובעים מתי לאפס את המערכת, מתי להפסיק את פעולת ה-Control ומהו תזמון השעון (מתי מתקבלת עליית שעון שלפיה כל המערכת עובדת) ובנוסף ישנו אות מוצא – done אשר מתריע מתי אנו סיימנו את פעילות התכנית.
אותות נוספים הם אותות ה-Control וה-Status כאשר אותות ה-Status הם אותות כניסה המתקבלים מ-OPC decoder והן אלו שמעדכנות את יח' ה-Control מהי הפקודה שעליו לבצע כעת ובהתאם לפקודה שנקבל ב-Control הוא משתמש באותות המוצא ממנו כדי לבצע את הפעולה כפי שהסברנו קודם.
**נציין כי לאחר עדכון נוספו עוד 3 אותות בקרה שמגיעים מה-control לצורך שליטה על MUX המשמשים להכנסת כתובות ל-Data Memory וMUX נוסף שבעזרתו קובעים כתובת קריאה/כתיבה מ-RF

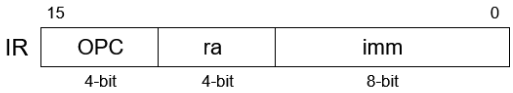



בתוך יחידת ה-Control בקוד שלנו ניתן לראות את כלל שלבי ביצוע פקודות, בדומה לאופן שלמדנו בקורס מבוא למחשבים שבקורס ראינו מהם אותות הבקרה שאותם היה צריך לשלוח כדי לבצע פקודה.
אופן ביצוע הפעולות התנהל כך :
תחילה המע' מתחילה במצב של Reset, כך שכל האותות מאופסים והאותות הם unaffected מלבד הצורך שלנו להתקדם להוראה הבאה לצורך ביצוע התכנית (תחילה אנו מתאחלים את כתובת ה-PC ל-0 ולאחר מכן אנו מקדמים את הכתובת ב-1).
לאחר מכן אנו עוברים ל-Decode שבשלב זה כבר התקבלה סוג ההוראה מתוך ה-OPC decoder, לכן אנו מתחילים בביצוע Decode לפי הפקודה שהתקבלה, שהרי אופן הביצוע משתנה לפי סוג הפקודה(Type) ובחלקן גם לפי הפקודה עצמה, כך שלמשל כלל פקודות R-Type ופקודות J-Type נבדלות באותות יחידים וכמעט כל התכנית זהה, זאת לעומת הפקודות St,Ld,Mov אשר לוקחות כמות מחזורים שונה ומאוד שונים מבחינת האותות אותם אנו צריכים להפעיל.
כאן התחלנו בפיצול הפקודות, זאת מכיוון שפקודות J-Type ופקודת Mov מסתיימות לאחר 2 מחזורים ושאר הפקודות ממשיכות לשלבי Execute באופן נפרד עבור R-Type ו-I-Type.



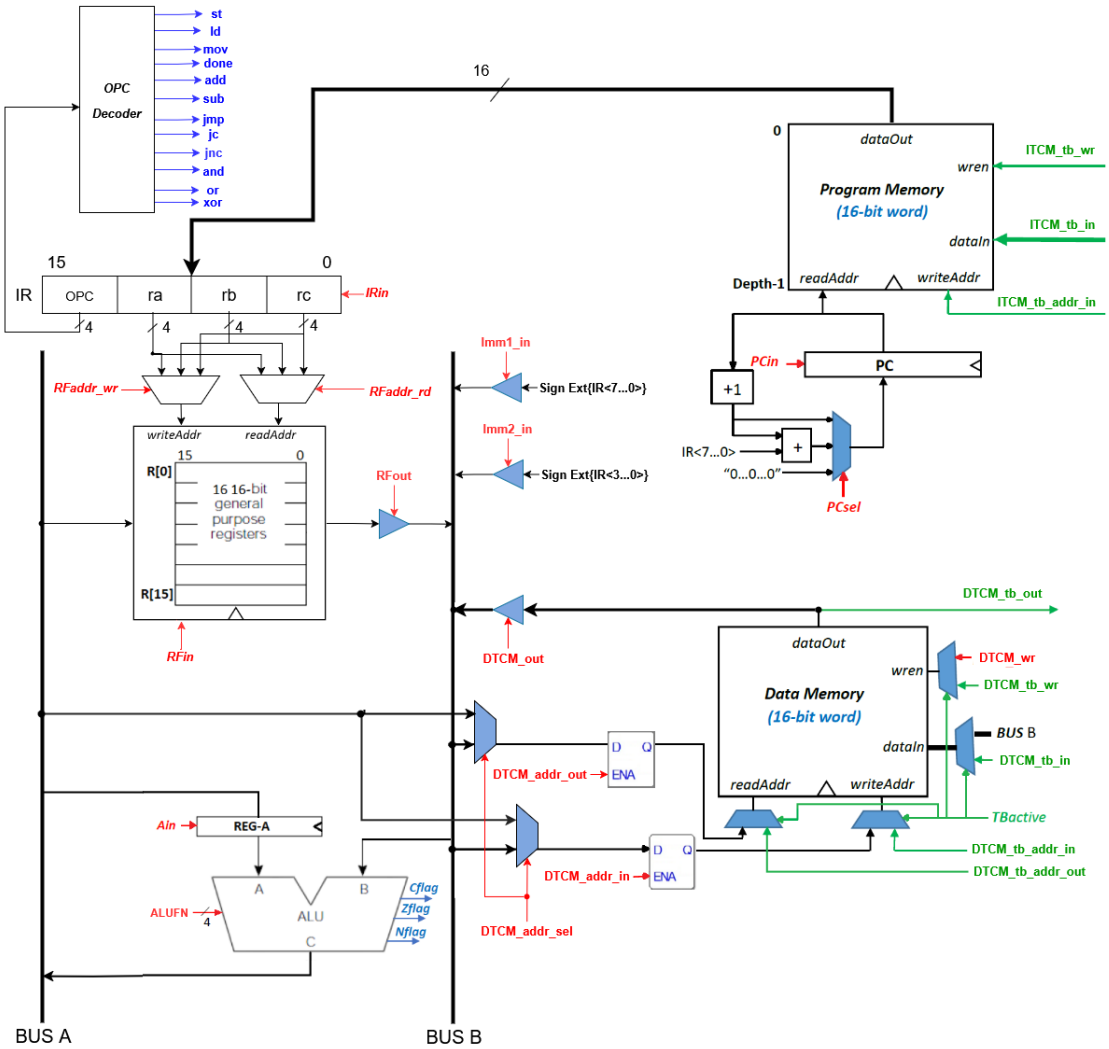



הרכיב השני שלנו הוא יח' ה-DataPath שבה נמצאים כלל הרכיבים שאנחנו מכירים ממבוא למחשבים – Decoder, PM(Program Memory), DM(Data Memory), IR, PC, RF, ALU
כאשר שלושת היחידות האחראיות על מידע (RF, DM, PM) התקבלו במטלה.
בכל המודולים שהיה עלינו לממש התחלנו ראשית מבנייתם בהתאם לדרישות המטלה, ביצוע TB נפרד בכדי לוודא את נכונות הרכיב ותפקוד בהתאם למה שאנו מצפים ורק לאחר מכן התקדמות לרכיב הבא.
לאחר סיום כל המודולים בכדי לוודא כי בנינו וחיברנו את הרכיבים כראוי, תחילה ביצענו חיבור בין שני רכיבים בלבד, ה-PC וה-PM ובנינו TB מתאים אשר בודק האם המוצא שאנו מקבלים תואם את הציפיות, לאחר מכן הוספנו את ה-IR והרחבנו את ה-TB, לאחר מכן הוספנו Decoder וכך המשכנו עד שהגענו למצב בו כלל המודולים מחוברים למערכת יחד עם Bidirpin, קביעת שני קווים לצורך העברת מידע (BUS-A, BUS-B), הכנסת מידע ע"י ITCM ו-DTCM ובנוסף הכנסת אותות בקרה מה-TB שיתפקד בתור Control בהתאם לפקודה הבאה שמתקבלת.

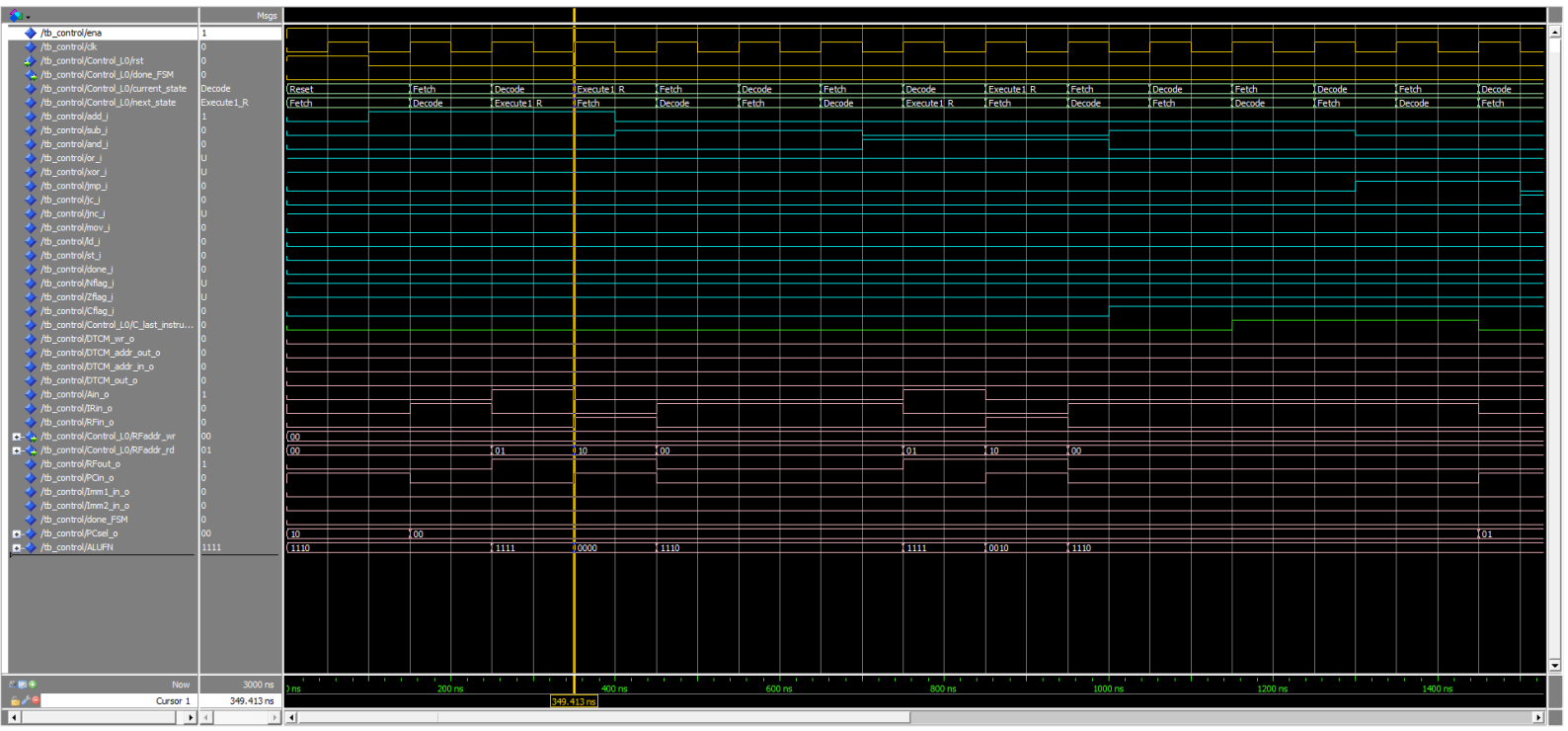
נציין כי ארכיטקטורת המעבד היא של שימוש בשני קווי מידע (BUS-A, BUS-B) מהצורה הבאה:



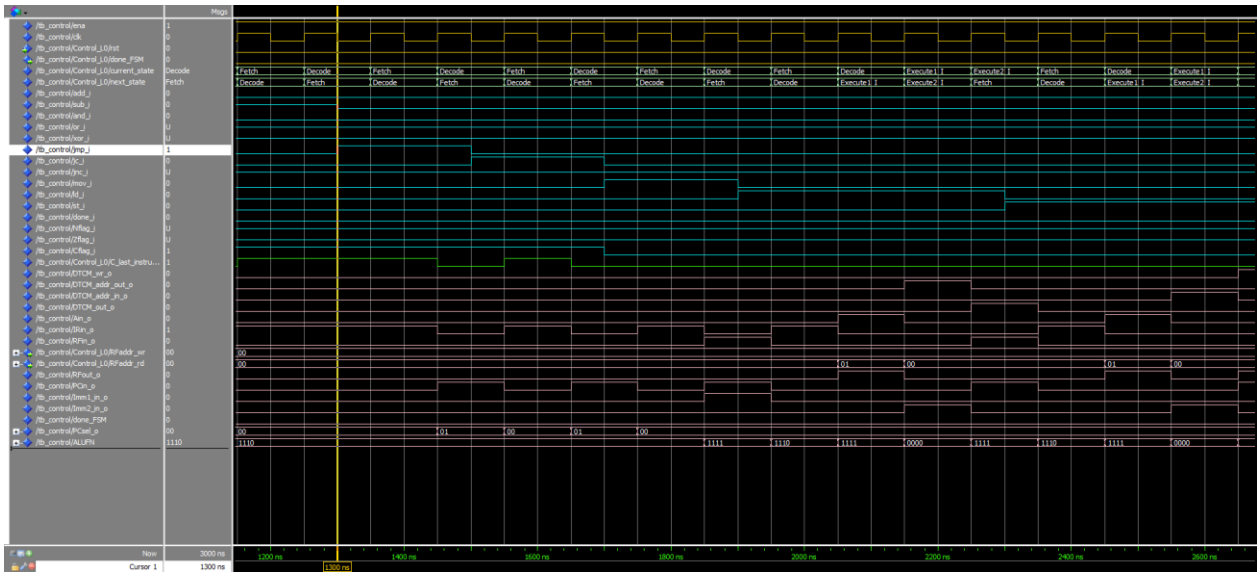

זאת בדומה לארכיטקטורת 2-BUS שאותה למדנו בקורס מבוא למחשבים ששם למדנו כיצד פועלים מעגלים מסוג זה.

כעת נציג את מימוש הקוד שלנו בעזרת הסיגנלים שקיבלנו בתכנית בעת הרצת ה-ModelSim.
בתור TestBench יצרנו קוד אשר מבצע Swap למערך בגודל 3 והדבר נעשה ב-TestBench של Datapath, Top ועבור ה-control אנו מבצעים בדיקה מה המוצאים עבור כל אחת מהפקודות לפי הסדר ב-TB של ה-Control.

<u>Control</u>:

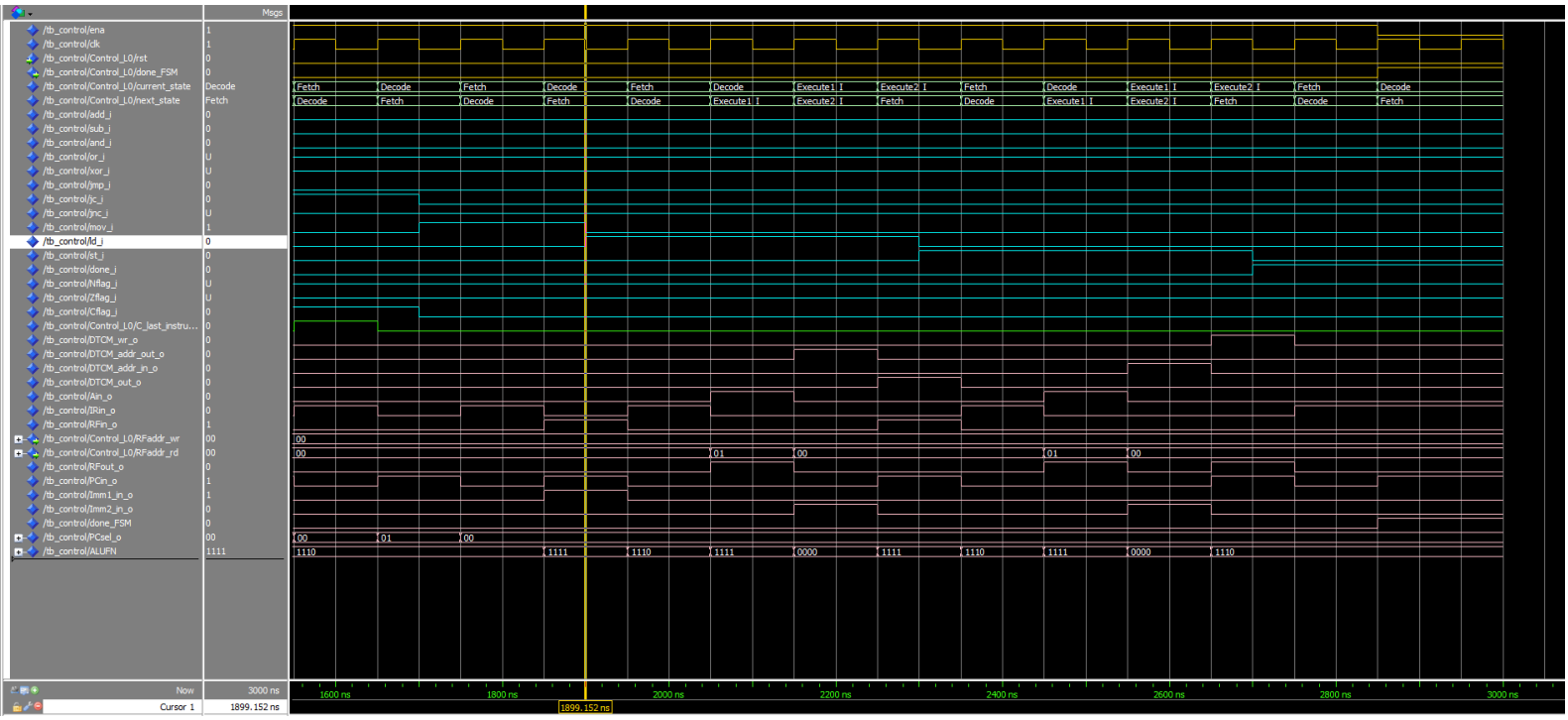



כפי שניתן לראות ראשית התחלנו מכניסת rst למשך 100 ns ולאחר מכן התחלנו בפעולות מסוג R-Type שכולן ממומשות באותו האופן ($Fetch \rightarrow Decode \rightarrow Execute1\_R$) ונבדלות אך ורק בפעולה שמבצע ה-$ALU$ בין כניסות $A$-$B$.
למשל כפי שניתן לראות ב-*350 ns* ניתן לראות כי אנחנו במחזור האחרון של פעולת $Add$ שבו מתבצע חיבור ב-$ALU$ ולכן ניתן לראות כי ה-$ALUFN$=*0000* שזהו ה-$OPC$ אשר משמש לחיבור ב-$ALU$ בנוסף אנו מכניסים את הערך החדש ל-$RF$ לרגיסטר *ra* ע"י $RFaddr_{wr} = 00$ בעזרת $RFin = 1$ ובמקביל אנחנו קוראים מ-*rb* בעזרת $RFaddr_{rd} = 01$ תוך ביצוע $RFout = 1$.
כמו כן, משום שאנו מסיימים פעולה אנו מבצעים *PCin=1*.



כעת אנו מתקדמים לפקודות *J-Type* ופקודות *I-Type* כאשר פקודות *J-Type* מתקיימות למשך 2 מחזורים ונבדלות בבדיקה האם התקבל *Carry* במחזור האחרון.

למשל בזמן *1300 ns* מתקבלת פקודת *Jmp* שלאחריה ניתן לראות כי $PCin = 1, PCsel = 01$ כך שאנחנו מוסיפים לכתובת של ה-*PC* את ה-*offset* המתקבל מפקודת *jmp*.

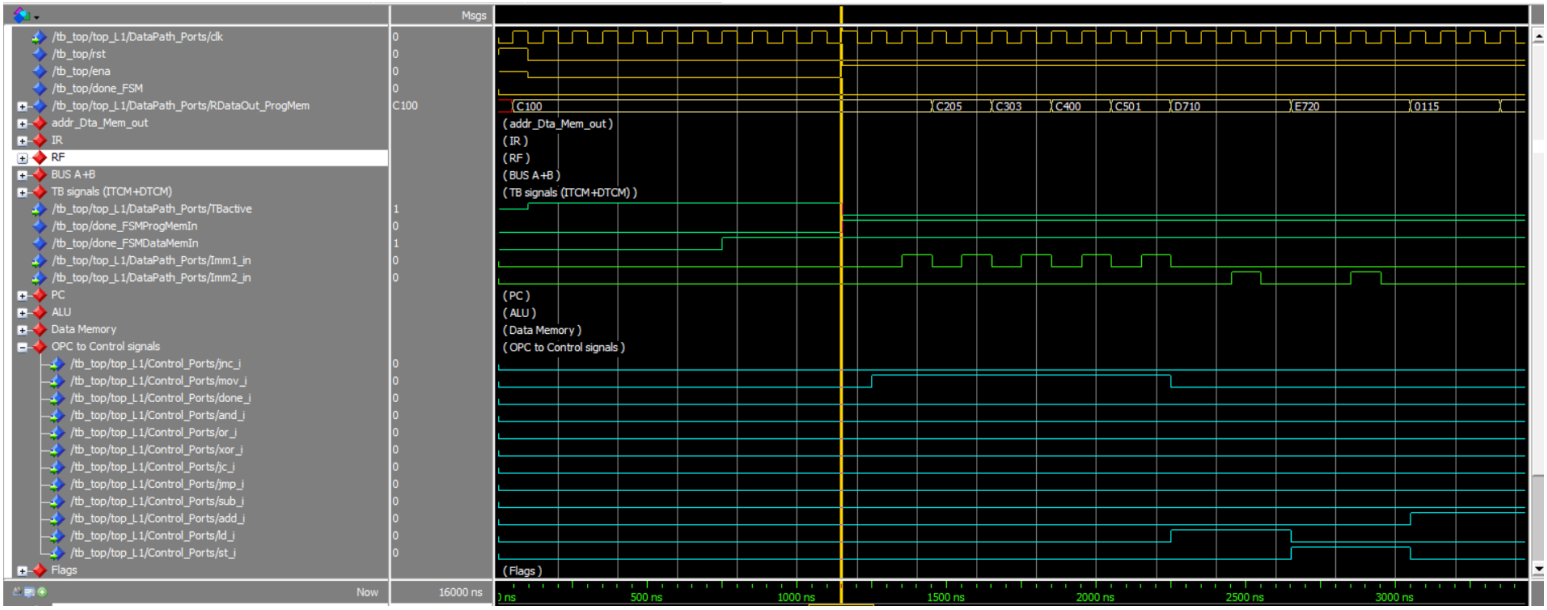


לאחר פקודות ה-J-Type הגענו לפעולות I-Type אנו לקראת סיום של פעולת mov בזמן 1900 ns שבה בדומה לקודם הכנסו לרגיסטר ra ערך ולכן ביצענו $ALUFN = 1111$ (C=B) והכנסה לתוך ה-RF ע"י $RFin = 1, RFaddr_{wr} = 00$ ונכנסים ל-Fetch של פקודת ld ולכן מבצעים $IRin = 1$.

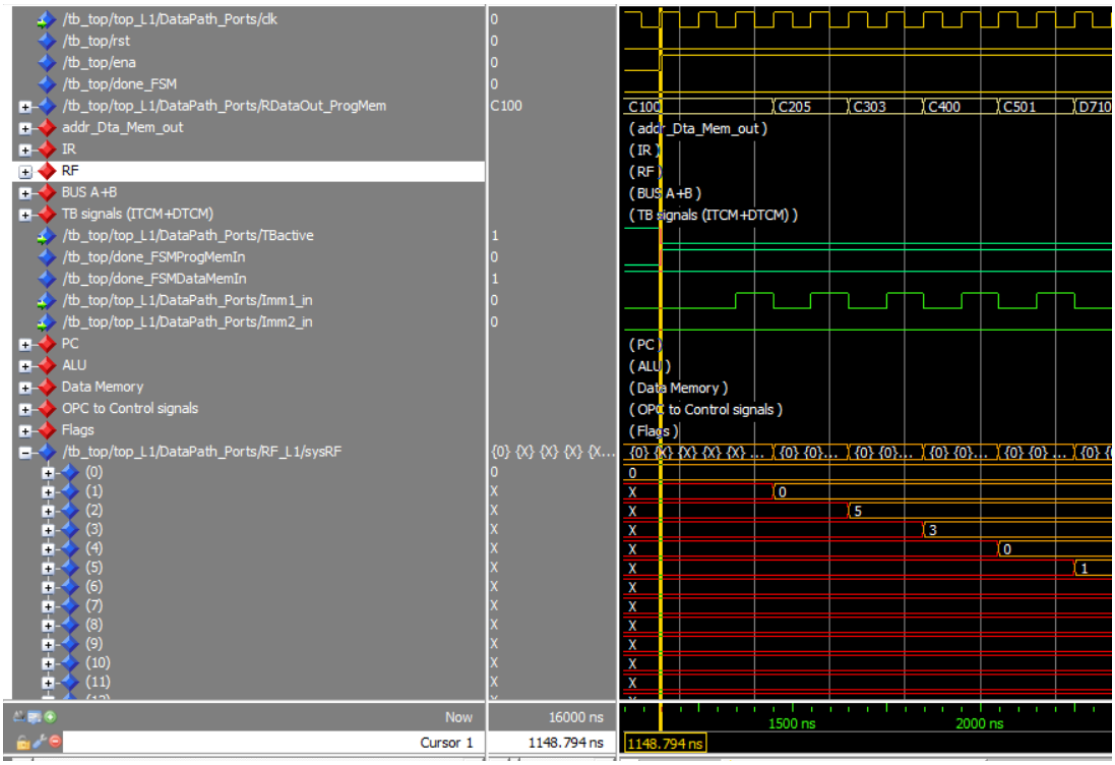
בדומה לקודם, משום שאנו מסיימים פעולה אנו מבצעים *PCin=1* במחזור האחרון בפקודת ה-*MOV*.

## *Top* :

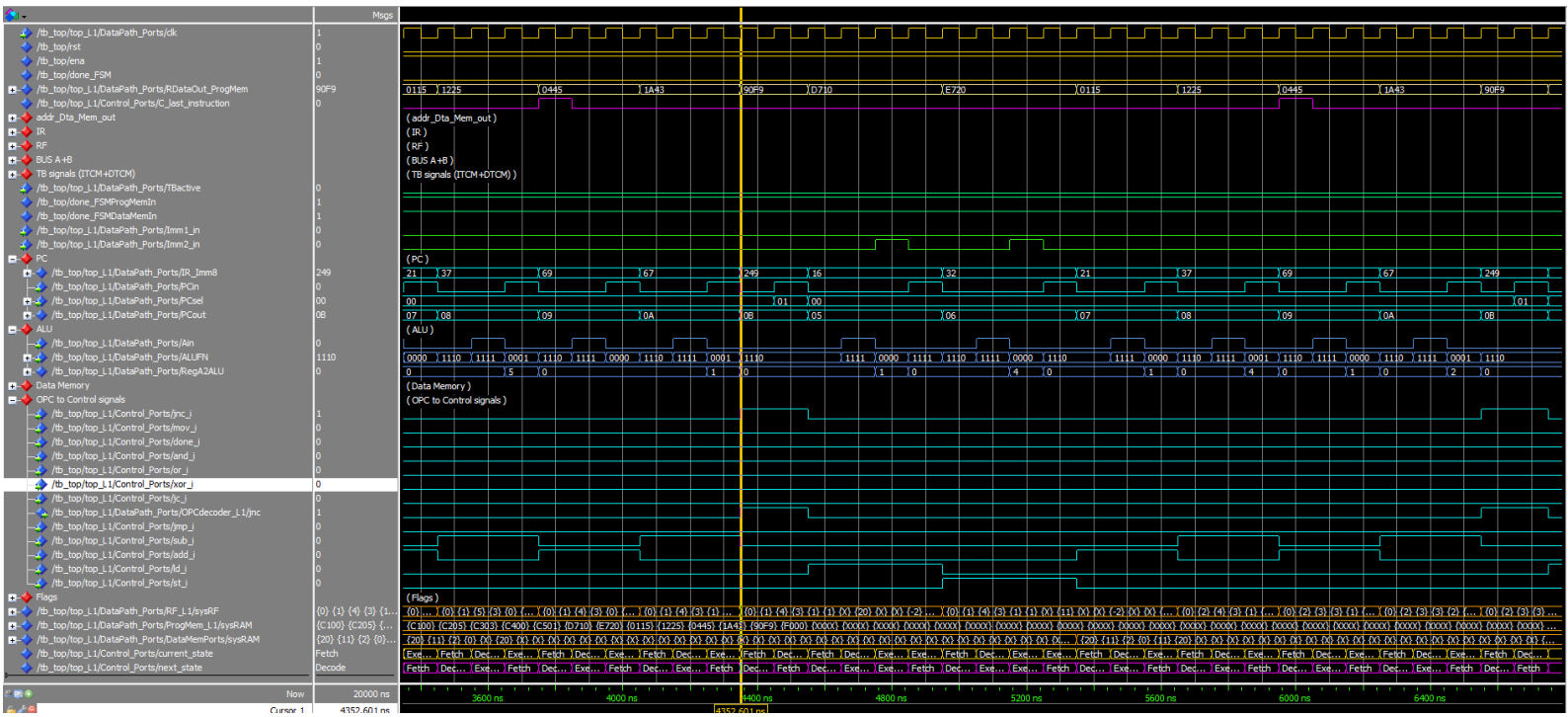
לצורך מעקב יותר נוח אחר גלי ה-*Top* וה-*DataPath* ביצענו קיבוץ לגלים לפי רכיב.
בקוד של ה-*Swap* התחלנו מהעברת מידע לתוך הרגיסטרים בעזרת פקודת *MOV* כפי שניתן לראות בתמונה.
ביצוע הפקודה מבחינת קווי בקרה זהה ל-*Control* כפי שראינו בתמונות למעלה אך כעת נראה כיצד קווי הבקרה משפיעים על זרימת המידע ב-*DataPath* :




בתמונה ניתן לראות כי תחילה אנו מתחילים מ-rst ולאחר מכן עוברים לאיתחול המערכת, זאת ע"י קריאת הזיכרון וקריאת ההוראות בעזרת PM ו-DM ורק לאחר מכן עוברים ל-Fetch היכן ש-TBactive יורד ל-0, ששם שמנו את המצביע ולאחר מחזור שעון אנו מקבלים כי הפקודה הינה MOV ומתבצעת 4 פעמים ולכן לוקח 8 מחזורי שעון עד לירידת קו ה-MOV.
בדומה לקווים שראינו קודם אנחנו נכניס את ערכי ה-Imm1 אל ה-RF במקום המתאים ולכן נראה לאחר פעולות אלו כי ב-RF התעדכנו הערכים בהתאם :

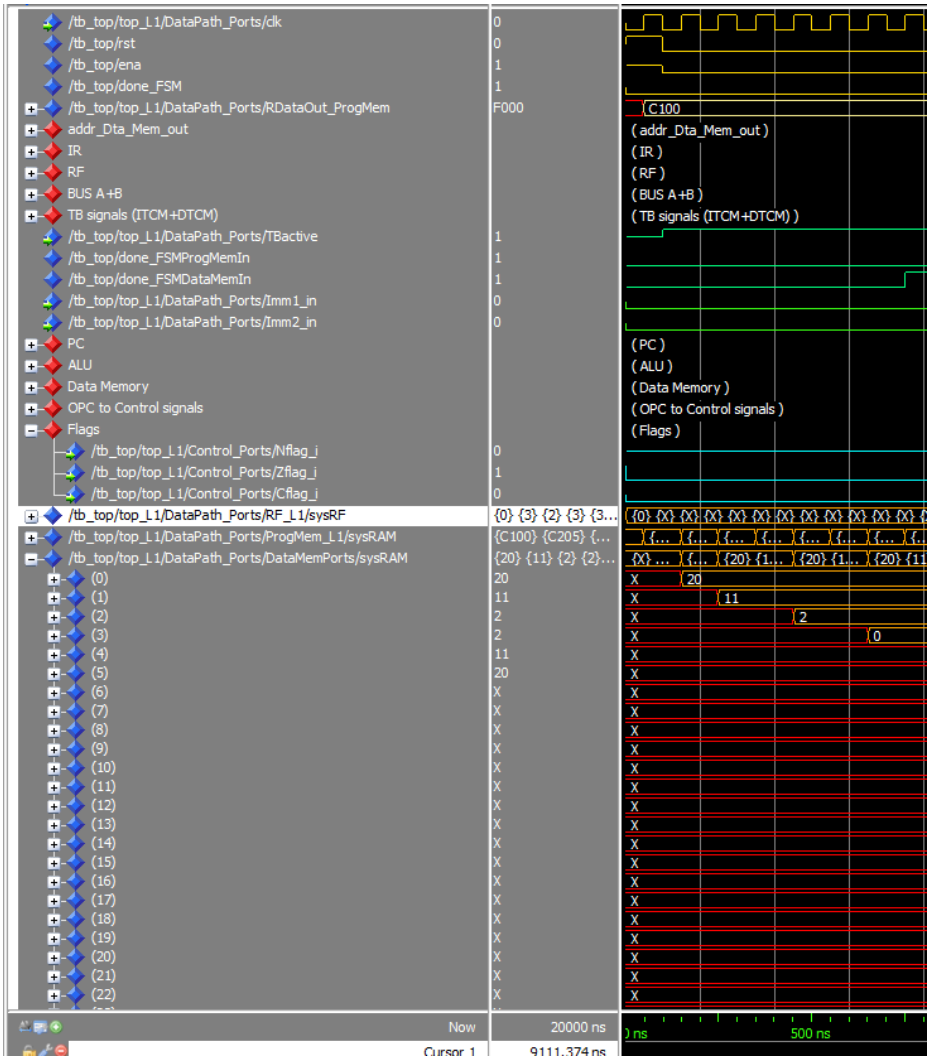


בתוך הקוד אנו מבצעים לולאה שרק לאחר ביצוע חיסור וקבלת carry אנחנו מסיימים, אחרת אנו חוזרים אחורה לבצע שוב את הלולאה כפי שרואים בתמונה שלא התקבל Carry:

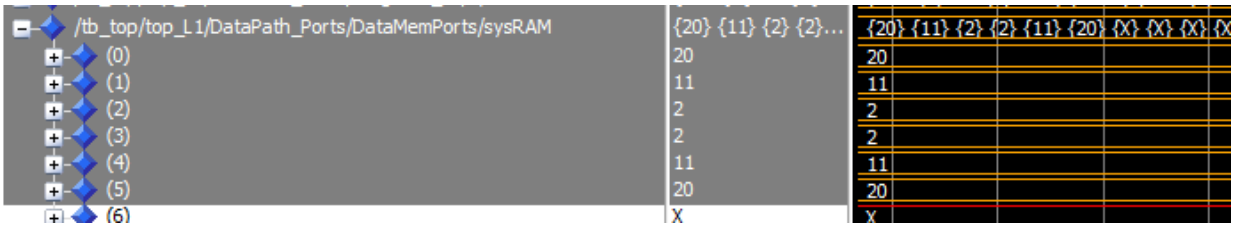


ניתן לראות בתמונה כי בעת ביצוע חיסור לא מתקבל carry מתוך C_last_instruction ולכן אנו חוזרים חזרה בקוד, אנו רואים זאת בפתיחת ה-PC תוך הכנסת ה-offset שלנו ולאחר JNC אנו חוזרים לפעולת ld שוב פעם אשר נמצאת בשורה 05 לאחר שהיינו לפני בשורה 0B.

אנו רואים כי  תחילה הזיכרון ב-Data Memory שלנו מכיל:




ובסיום ביצוע הפעולות קיבלנו כי הזיכרון הוא:




כלומר הקוד אכן עבד וביצע SWAP וכתב אותו בזיכרון כמבוקש.

DataPath:

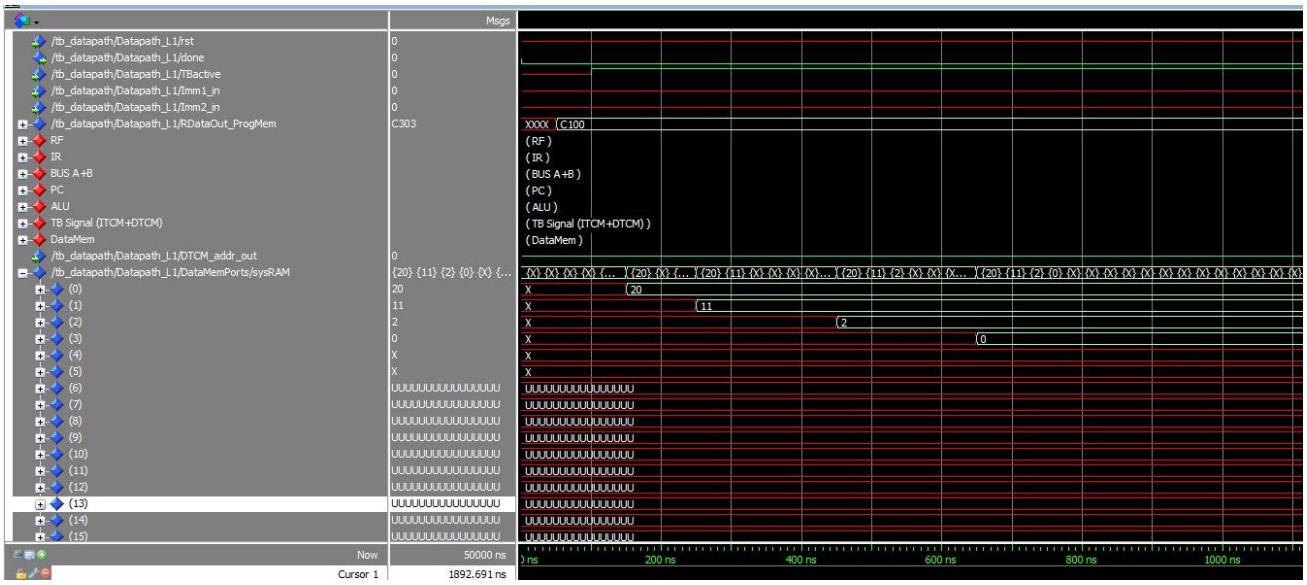
בדומה ל-Top אנו מבצעים את אותו הקוד אך כעת רק על יחידת ה-DataPath ולכן נצפה לראות התנהגות דומה.

ואכן ניתן לראות בפקודות ה-MOV כי אנחנו שוב מכניסים ערכים אל ה-RF ואורך הפקודה זהה למה שהיה ב-top, אך נציין כי לצורך ביצוע הפעולה כתבנו ידנית את רצף הפקודות והדבר לא נעשה באופן "אוטומטי" כפי שקרה בעת הפעלת ה-top, זאת משום שכעת אין לנו "מוח" שמבצע את הפעולות ולכן הכל נעשה מתוך ה-Testbench.

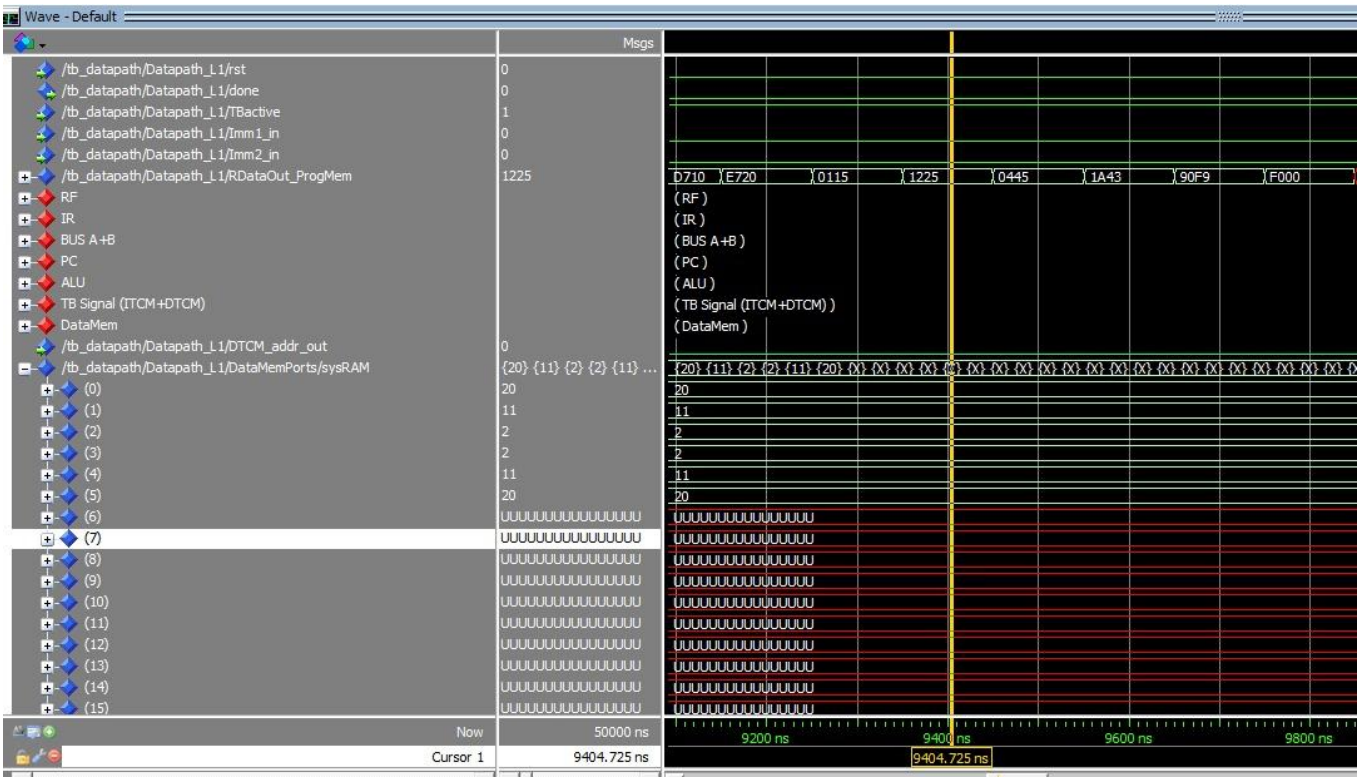
למשל בתמונה המצורפת רואים כי בנק' המסומנת לאחר סיום הפקודה $0xC205$ אנו מתבצע MOV של 5 ל-r2.

בנוסף אנו רואים כי בתחילת הקוד המידע שנטען אל הזיכרון זהה למה שנטען קודם וכך גם הזיכרון בסיום הפעולות, הדבר מעיד על כך שריצת ה-TestBench עברה באופן תקין וכנדרש.

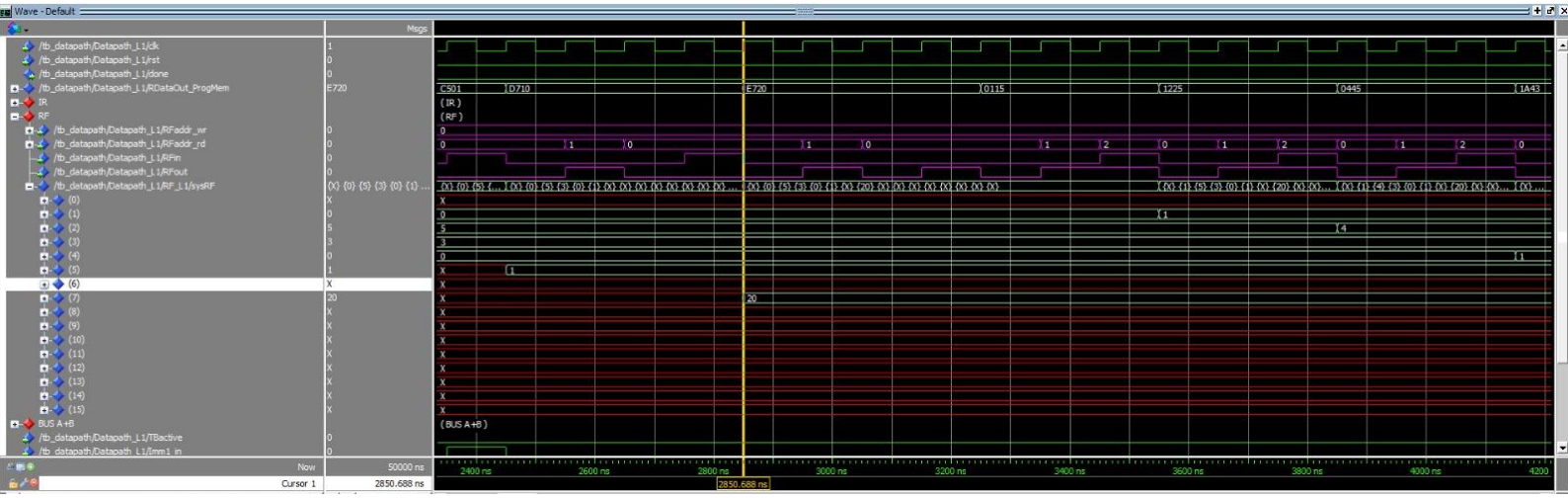
בתחלה:




סיום כך שניתן לראות גם את פקודת ה-Done שבה אנו כותבים לקובץ טקסט עם מידע מעודכן בסוף התכנית:



כמו כן, צירפנו תמונה ובה ניתן לראות את הפעילות ב-DataPath בעת ביצוע פעולת Ld:




ניתן לראות כי נרשם לנו לתוך r7 הערך מהזיכרון שמיוצג ע"י $Mem[R[r1]+0]$ ולכן התקבל הערך 20.

העבודה הייתה מאוד מעניינת ושמחנו לראות כיצד לממש כלים ממס' קורסים בעבודה אחת!