Preparation Report LAB3

ADVANCED CPU ARCHITECTURE AND HARDWARE ACCELERATORS LAB

361.1.4693

Nachman Mimoun 321730558

Danel Barsheshet 209471242

מבוא

במעבדה זו, מטרתנו הייתה לפתח תכנון בסיסי של מעבד מסוג multi-cycle הכולל יחידת שליטה (Control Unit) לצד נתיב נתונים(Datapath) . המעבד תוכנן לביצוע מספר פקודות יסודיות המפורטות ב ISA-המצורפת.

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]	-	-	-
	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

Table 1 : Multi-cycle CPU ISA

מימוש יחידת הבקרה התבצע באמצעות מכונת מצבים סופית (FSM) מסוג Mealy, בהתאם לנלמד בקורס המקביל. נתיב הנתונים (Datapath) מיושם באופן מקבילי, כך שבהינתן אותות הבקרה המגיעים מיחידת הבקרה, היחידה מבצעת את הנדרש ב-cycle הנוכחי של הפקודה. להלן תיאור המערכת אותה נרצה לממש:

3. Controller based system:

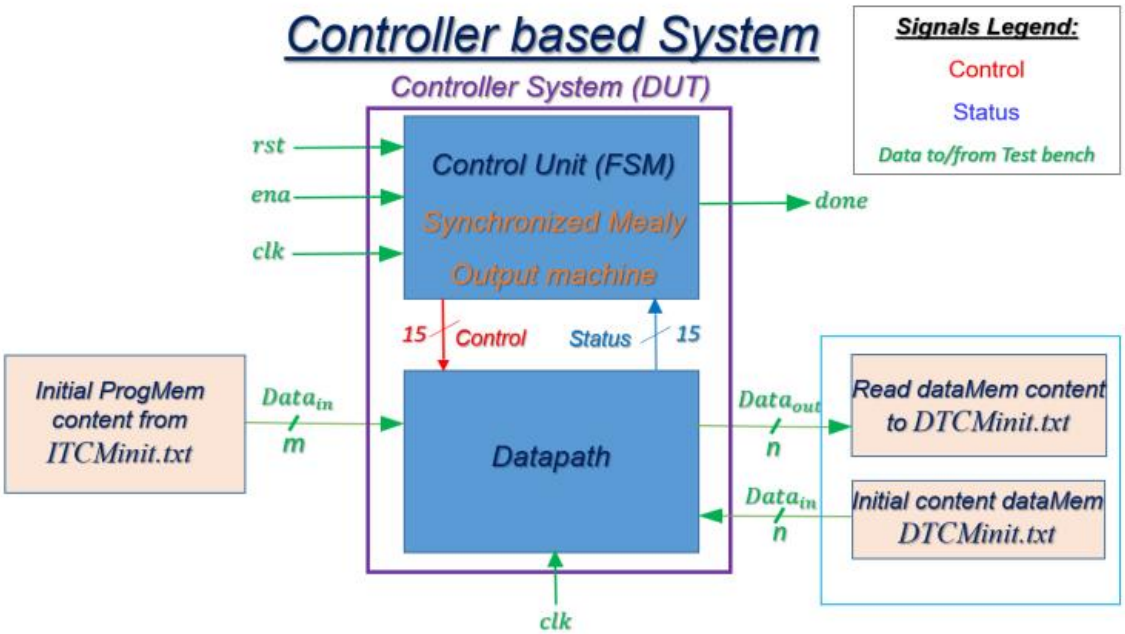


Figure 1: Overall DUT structure

המערכת שלנו תקבל שני קבצי טקסט המשפיעים על פעילות המעבד, בנוסף לאותות בקרה שונים:

- init_ITCM.txt קובץ המכיל את כל ההוראות של התוכנית אותה נרצה להריץ, כאשר הפקודות כתובות בפורמט הקסדצימלי. קובץ זה מאתחל את זיכרון התוכנית(Program Memory).
- init_DTCM.txt קובץ המכיל את כל המידע שיאוחסן בתאי הזיכרון לפי הסדר - כלומר, בתא 0 יאוחסן הערך מהשורה הראשונה בקובץ וכן הלאה. קובץ זה מאתחל את זיכרון הנתונים(Date Memory).
- rst, clk, ena, done

בסיום הרצת התוכנית, המערכת תייצר קובץ טקסט המכיל את תוכן זיכרון הנתונים (DataMem) של המערכת בתום התוכנית, יחד עם דגל סיום(Done).

Control Unit

דיאגרמת בלוק –

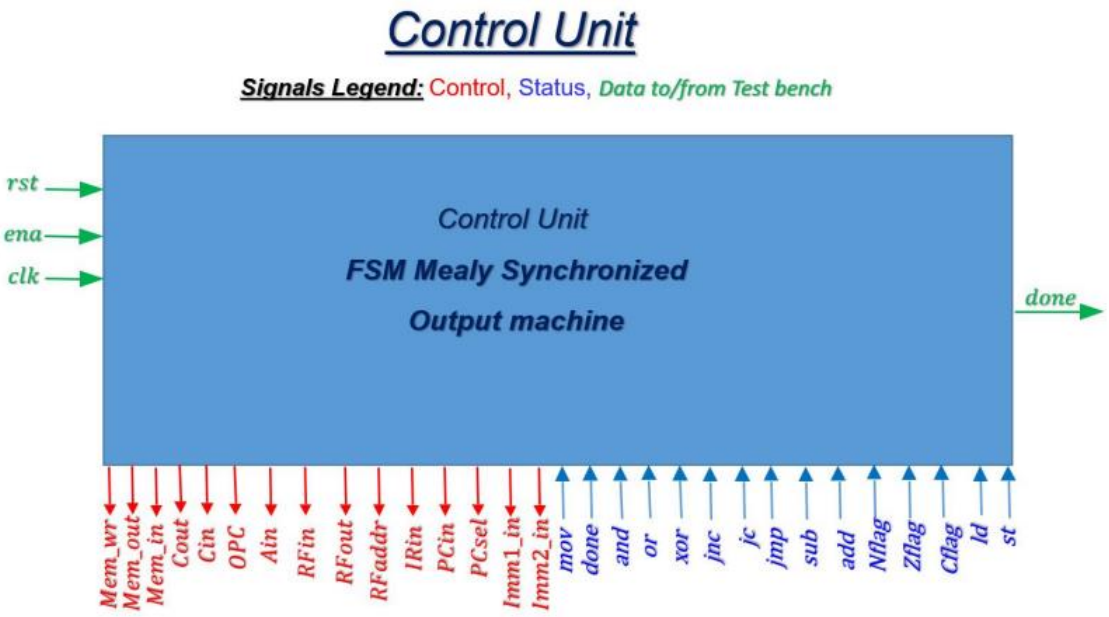


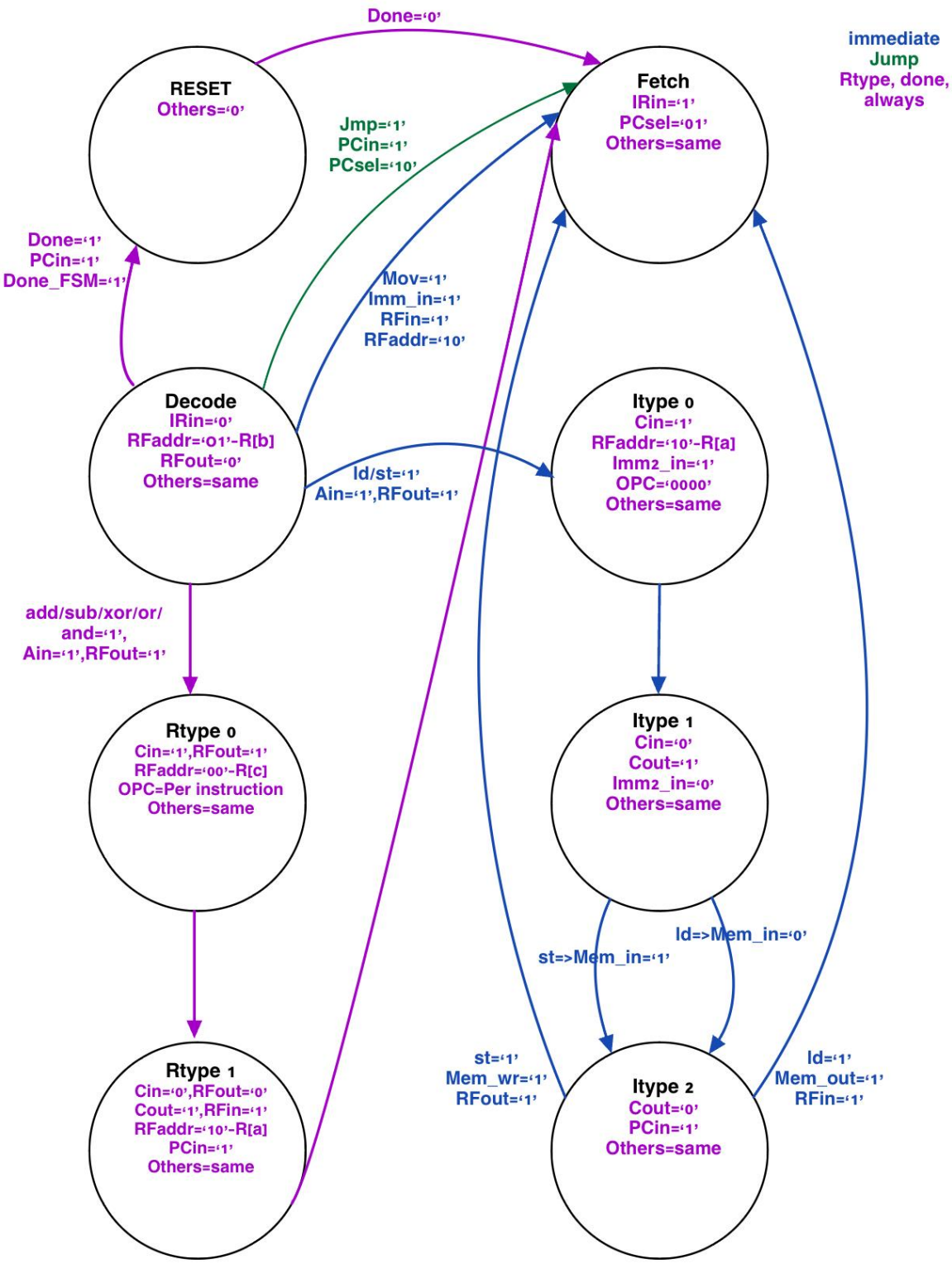
Figure 4: Control unit structure

אותות הבקרה היוצאים לעבר נתיב הנתונים(Datapath) מסומנים באדום.
אותות ה-Status שה-Datapath שולח ליחידת השליטה(Control Unit), המעבירים מידע על הפקודה המתבצעת לאחר פענוח הפקודה מה- IR מסומנים בכחול.
אותות ה TB-מסומנים בירוק.

תיאור המודול –

ה- Control Unit היא החלק במערכת המקביל למוח. תפקידה לקבל את דגל הפקודה אותה יש לבצע הנגזר ישירות מרגיסטר ההוראה IR - ובהתאם לדגל זה, להפעיל אותות בקרה מתאימים על פי ה FSM-המצורף. אותות הבקרה הללו מועברים ליחידת נתיב הנתונים(Datapath) ומכתיבים את אופן פעולתה.

בהתאם ליחידות ה -BUS וה-ISA המצורפת, על יחידת הבקרה לבצע את הוראותיה במספר מחזורים רצופים, כמתואר ב FSM-הבא:



ה-FSM כולל מספר סוגי פקודות, בין היתר כתוצאה מהסוגים השונים של ההוראות המפורטות ב-ISA הוראות מסוג: R-Type, J- Type, I- Type לשם כך, חילקנו את ה-FSM הן למחזורים המשותפים לכל הפקודות (Fetch, Decode) והן למחזורים המתפצלים בהתאם לסוג הפקודה המתקבלת:

- Fetch חישוב כתובת הפקודה הבאה לביצוע וקריאתה מהזיכרון.
- Decode פענוח הפקודה בנתיב הנתונים(Datapath) והעברתה ליחידת הבקרה, שבהתאם לכך יודעת איזו הוראה לבצע ואילו דגלים להפעיל. בנוסף, הבאת ערכי הרגיסטרים בהם הפקודה משתמשת בחלק מהפקודות.
- R-type פקודות בעלות המבנה OP|Ra|Rb|Rc
- J-type פקודות בעלות המבנה OP|0000|offset
- I-type פקודות בעלות המבנה OP|Ra|Rb|imm או OP|Ra|imm

Datapath Unit

דיאגרמת בלוק –

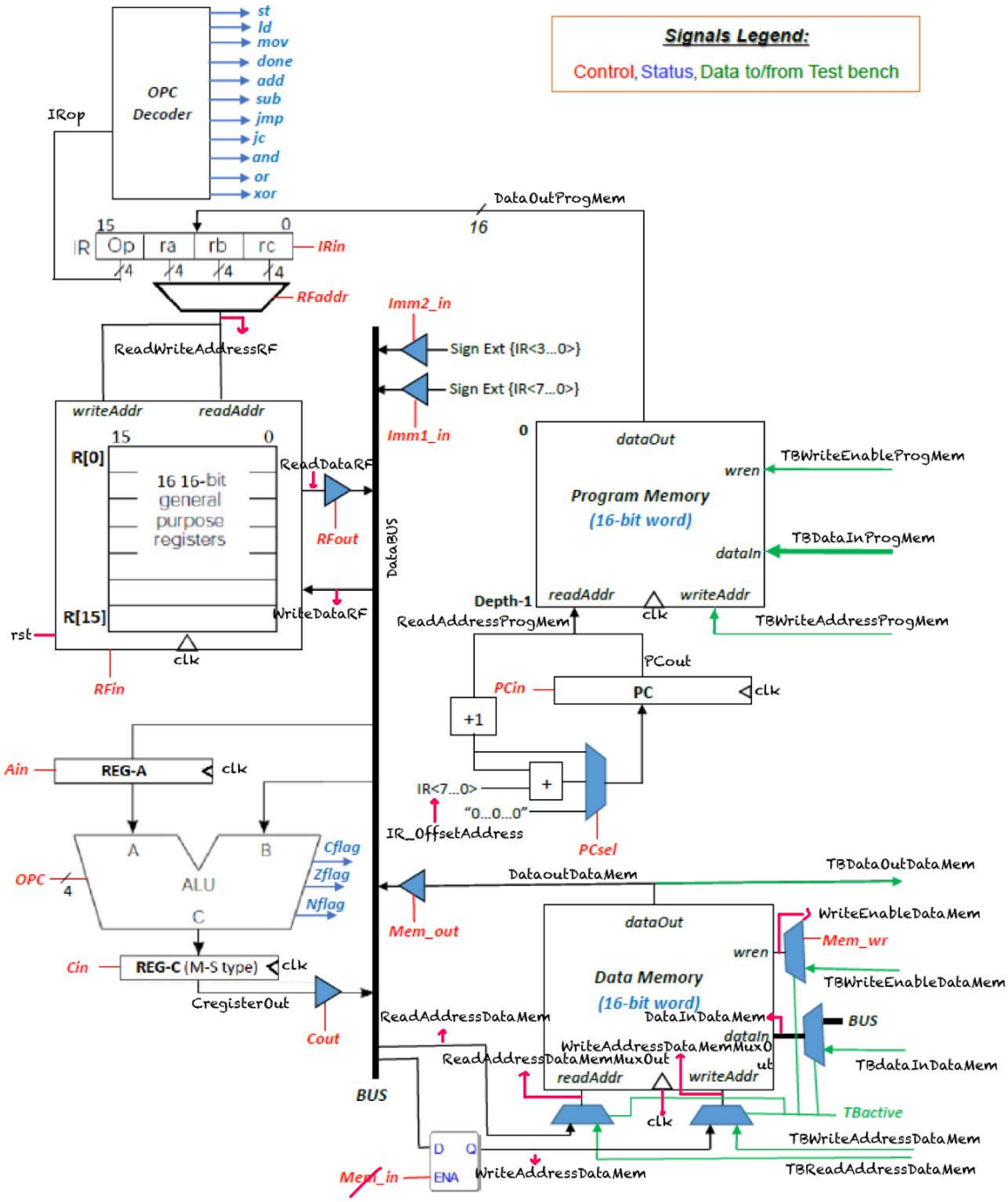


Figure 2: Datapath structure

תיאור המודול –

יחידת נתיב הנתונים (Datapath Unit) היא החלק במערכת שניתן להשוות לשרירים של המערכת. תפקידה העיקרי הוא לקבל את הפקודה הרצויה מזיכרון התוכנית (ProgMem) ולחלץ ממנה, באמצעות מפענח (decoder), את דגל הסטטוס של הפקודה שיש לבצע ולהעבירו ליחידת הבקרה. בנוסף, מטרתה המרכזית של יחידה זו היא לבצע את המודולים הסינכרוניים והא-סינכרוניים הנדרשים לפעולת הפקודה הנדרשת, בהתאם לאותות הבקרה המתקבלים.

נדגים את התהליך באמצעות אחת הפקודות. נניח שברצוננו לבצע את הפקודה הבאה: D208. כאשר נקרא נתון זה אל ה-ProgMem, נקבל את הקידוד הבינארי הבא: 1101001000001000. מכאן נוכל לחלץ את הפורמט של ההוראה הנדרשת:

$$ld\ r_2, imm(r_0): R[r_2] \leftarrow M[8 + R[r_0]]$$

תהליך ביצוע הפקודה:

1. **שלב ה-Fetch**: נביא את הפקודה הנדרשת אל תוך רגיסטר ההוראה (IR). כתוצאה מכך, יועלה דגל הסטטוס המתאים המציין איזו פקודה יש לבצע (במקרה זה, 'ld = '1).

2. **שלב ה-Decode**: נרצה להוציא ל-BUS את ערך רגיסטר r_b, שבמקרה שלנו יהיה 0 (r0 מאותחל ל-0 בהתחלה נשים לב כי לפי הקוד של RF רק רגיסטר 0 מאופס בהתחלה, כלומר לא ניתן יהיה לקרוא נתון מרגיסטר אחר שהוא לא 0 בתוכנית לפני שכתבנו אליו משהו). נעלה את RFaddr = "01" ו-'RFout = '1 כדי להוציא את הכתובת המתאימה ל-RF ואז להוציא את הערך של R[r_b] לBUS. במקביל, נפעיל 'Ain = '1 כדי ש-A יקבל את הערך שבBUS במחזור הבא(REG-A קורה בעליית שעון ומתעדכן בסוף ה-Process).

3. **שלב ItypeState_0**: נבצע את החיבור של A עם ערך Imm כדי למצוא את הכתובת ממנה נרצה לקרוא בזיכרון הנתונים. נוציא את Imm2 לבאס עם 'Imm2_in = '1, וניתן ל-ALU פקודת חיבור עם "OPC = "0000. התוצאה תהיה R[r_b] + Imm. נפעיל 'Cin = '1 כדי שבמחזור הבא מוצא C-REG יחזיק את תוצאת החיבור. בנוסף, נשנה את כתובת הרגיסטר של RF שנרצה לכתוב אליו ל-r_a, כלומר "RFaddr = "10.

4. **שלב ItypeState_1**: נוציא את תוצאת החיבור לכתובת ממנה נרצה לקרוא ב-DataMem. נפעיל 'Cout = '1 כדי להוציא את התוצאה (מוצא C-REG) לבאס. חשוב לוודא ש- 'Imm2_in = '0 כדי למנוע התנגשות בבאס.

5. **שלב ItypeState_2**: נוציא את הערך בזיכרון בכתובת שחישבנו (R[r_b] + Imm). נפעיל 'Mem_out='1 להוצאת המידע לבאס (תוך הקפדה על 'Cout = '0 למניעת התנגשות). נכתוב לרגיסטר R[r_a] על ידי הפעלת 'RFin = '1 ווידוא ש-"RFaddr = "10. לבסוף, נקדם את מונה התוכנית (PC) עם 'PCin='1 ונחזור לשלב ה-Fetch.

תהליך זה מדגים את האופן בו נתיב הנתונים(Datapath) מבצע את הפעולות הנדרשות בכל שלב של ביצוע הפקודה, תוך שימוש באותות הבקרה המתאימים ובמרכיבי החומרה השונים של המעבד.

תוצאות סימולציה

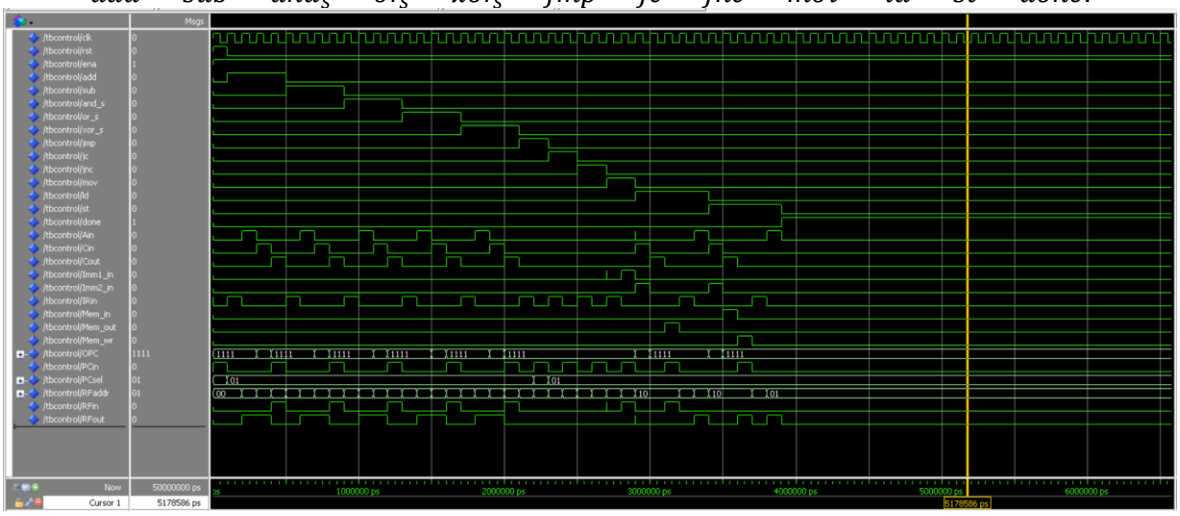
נחלק את תוצאות הסימולציה לשלושה חלקים:

- סימולציה עבור יחידת השליטה (Control Unit)
- סימולציה עבור נתיב הנתונים (Datapath Unit)
- סימולציה עבור המערכת כולה

סימולציה עבור ה-Control Unit

במהלך סימולציה זו, נפעיל דגלים של פקודות שונות שנתיב הנתונים(Datapath) אמור להעביר ליחידת השליטה(Control Unit) לאחר ביצוע הפענוח (decode). לאחר מכן, נבדוק שאותות הבקרה המופעלים בכל אחד מהמחזורים תואמים את הציפיות שלנו. עבור כל פקודה, נפעיל את הדגל למשך מספר המחזורים הנדרש לביצועה.
להלן תוצאת הסימולציה המלאה, כאשר סדר הפקודות הוא:

$$add \rightarrow sub \rightarrow and_s \rightarrow or_s \rightarrow xor_s \rightarrow jmp \rightarrow jc \rightarrow jnc \rightarrow mov \rightarrow ld \rightarrow st \rightarrow done.$$

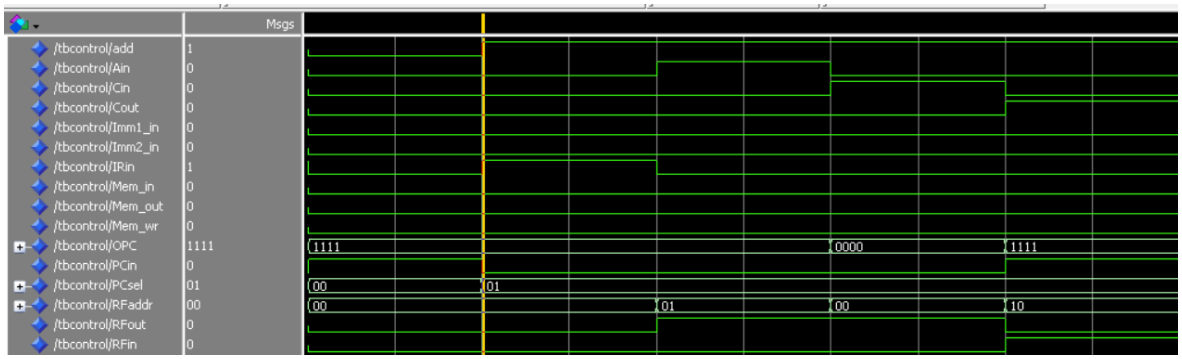


כעת נסקור כל פקודה בנפרד ונוודא שהדגלים המופעלים אכן תואמים את המצופה לפי ה- FSM המצורף.

פקודת add –

בפקודת add נצפה שנעבור דרך מצבי ה- FSM הבאים: fetch → decode → Rtype_0 → Rtype_1

סימולציית הגלים –



ניתן לראות שפקודת ה- add מכילה 4 מחזורים(cycles):

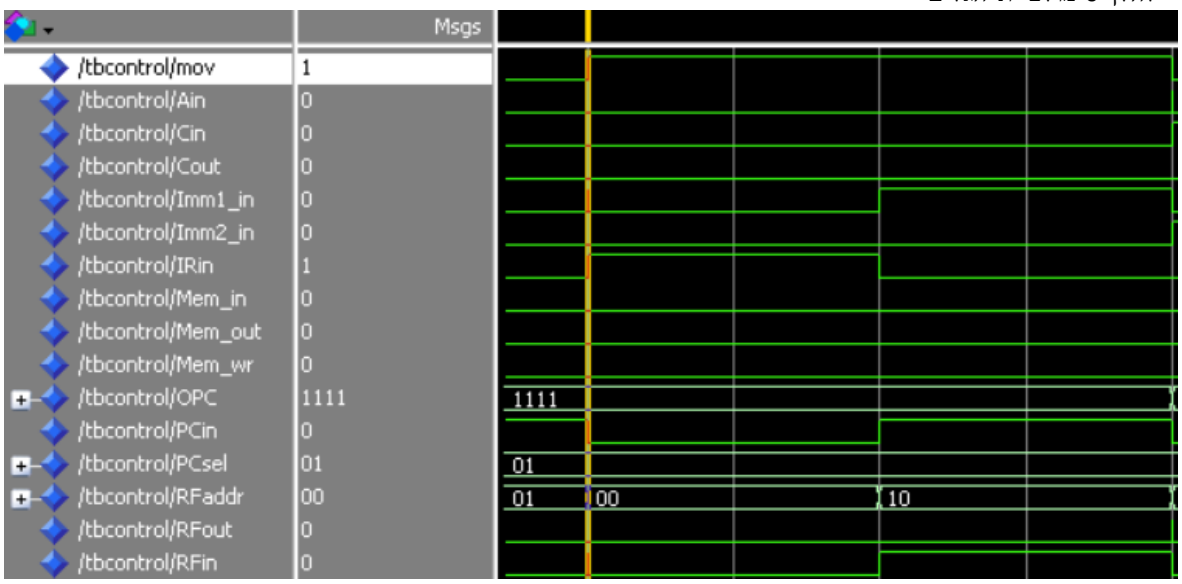
1. במחזור הראשון IRin, עולה כתוצאה מהגדרת ה- fetch במהלכו נרצה להכניס ל- IR את ערך הפקודה לביצוע.
2. לאחר מכן מתחיל שלב ה-decode במהלכו נרצה להחזיר את הדגל המתאים. במקביל, ננצל את ה- BUS כדי להכניס ערך לרגיסטר A כחלק מהגדרת פעולת add.
3. במחזור השלישי, נשים על ה- BUS את הרגיסטר השני כדי שייכנס לכניסת B של ה- ALU ויתבצע החישוב (OPC = "0000") מסמן חיבור כמו כן, נפעיל את דגל Cin כדי שתוצאת החישוב תיכנס לרגיסטר C.

4. במחזור האחרון, נוציא את תוצאת החישוב המוחזקת ב- C אל ה- BUS כדי שתיכנס לרגיסטר דרך ה- RF ולכן נפעיל את דגל Cout.

פקודת mov

בפקודת mov נצפה שנעבור דרך מצבי ה- FSM הבאים: fetch → decode.

להלן סימולציית הגלים –



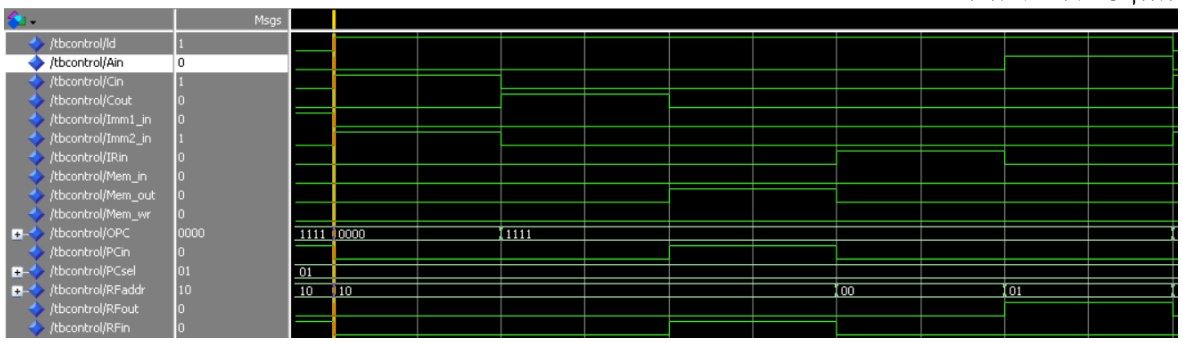
ניתן לראות שפקודת ה- mov מכילה 2 מחזורים:

1. במחזור הראשון IRin, עולה כתוצאה מהגדרת ה- fetch במהלכו נרצה להכניס ל- IR את ערך הפקודה לביצוע.
2. במחזור השני מתחיל שלב ה- decode אם דגל הפקודה שעלה הוא mov נעביר את הערך מה- Immediate אל ה- RF לכן, נפעיל את Imm1 כדי להכניס את המספר מה- Immediate ל- BUS וכן נפעיל את RFin כדי להכניס את המידע מה- BUS אל הרגיסטר המתאים שנקבע ב- RFaddr.

פקודת ld

בפקודת ld נצפה שנעבור דרך מצבי ה- FSM הבאים fetch → decode → Itype_0 → Itype_1 → Itype_2 :

להלן סימולציית הגלים –



פקודה זו מבצעת:

$$ld\ r_a, imm(r_b): R[r_a] \leftarrow M[imm + R[r_b]]$$

ניתן לראות שפקודת ה ld-מכילה 5 מחזורים:

1. שלב ה- Fetch:
   - נביא את הפקודה הדרושה אל תוך רגיסטר ההוראה (IR).
   - יופעל דגל הסטטוס המתאים, במקרה זה 'ld='1.
2. שלב ה- Decode:
   - נוציא ל-BUS את ערך רגיסטר r_b (במקרה שלנו יהיה 0, כי r_0 מאותחל ל-0 בהתחלה).
   - נפעיל "RFaddr="01 ו- 'RFout='1 כדי להוציא את הכתובת המתאימה ל- RF ואז להוציא את הערך של R[r_b] ל-BUS.
   - נפעיל 'Ain='1 כדי שבמחזור הבא A יקבל את הערך שב-BUS (A-REG מתעדכן בעליית שעון בסוף ה-process).
3. שלב ItypeState0:
   - נבצע את החיבור של A עם ערך Imm כדי למצוא את הכתובת שנרצה לקרוא מזיכרון ה-Data.
   - נוציא את Imm2 לבאס עם 'Imm2_in='1.
   - ניתן ל- ALU פקודת חיבור עם "OPC="0000.
   - תוצאת החיבור תהיה R[r_b] + Imm.
   - נפעיל 'Cin='1 כדי שבמחזור הבא מוצא C-REG יחזיק את תוצאת החיבור.
   - נשנה את כתובת הרגיסטר של RF שנרצה לכתוב אליו ל- r_a כלומר, "RFaddr="10.
4. שלב ItypeState1:
   - נוציא את תוצאת החיבור לכתובת שנרצה לקרוא ממנה ב-DataMem.
   - נוציא את התוצאה (מוצא C-REG) לBUS עם 'Cout='1.
   - נוודא ש- 'Imm2_in='0 כדי למנוע התנגשות בבאס.
5. שלב ItypeState2:
   - נוציא את הערך בזיכרון בכתובת שחישבנו (R[r_b] + Imm).
   - נוציא את המידע לבאס עם 'Mem_out='1 (ונוודא 'Cout='0 כדי למנוע התנגשות).
   - נכתוב לרגיסטר R[r_a] על ידי הפעלת 'RFin='1 ווידוא ש-"RFaddr="10.
   - נקדם את מונה התוכנית (PC) עם 'PCin='1.
   - נחזור לשלב ה-Fetch.

סימולציה זו מדגימה את הפעלת אותות הבקרה הנכונים בכל שלב של ביצוע פקודת ld, תוך שמירה על סדר הפעולות הנכון ומניעת התנגשויות על ה-BUS. היא מראה כיצד יחידת השליטה(Control Unit) מנהלת את זרימת המידע והפעולות בנתיב הנתונים(Datapath) לביצוע מדויק של הפקודה.

סימולציה עבור ה-Datapath Unit

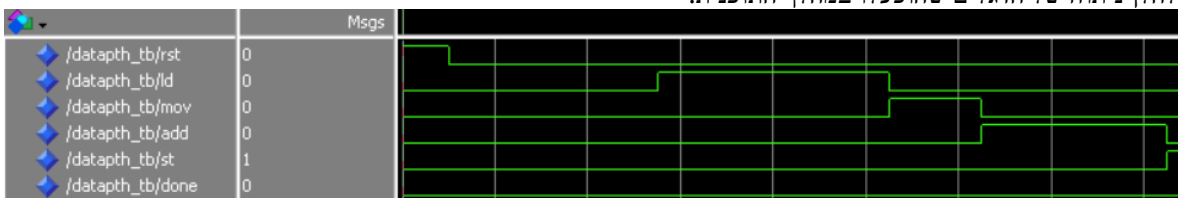
בסימולציה זו, השתמשנו בפונקציית report בנתיב הנתונים עצמו כדי להדפיס את האותות וקווי הבקרה המתאימים. זה מאפשר לנו לדבג בצורה יעילה יותר, כך שנוכל לראות גם את ערכי האותות וגם את תוכן הזיכרונות - הן את זיכרון הנתונים (DataMem) והן את קובץ הרגיסטרים(Regfile).

בסימולציה זו נפעיל את אותות הבקרה המתאימים לפי ה- FSM של יחידת הבקרה(Control Unit), אשר אמורים להיכנס לנתיב הנתונים(Datapath), ונבדוק האם התקבלו ערכים תקינים כפי שהיינו מצפים מכל פעולה.

בבדיקה זו (Test Bench) נבצע את הפעולות הבאות:

$$ld\ r_3 \leftarrow M[2+r_0], Mov\ r_2 \leftarrow 5, Add\ r_4 \leftarrow r_3 + r_2, st\ M[1+r_0] \leftarrow r_4$$

להלן ניתוח של הדגלים שהופעלו במהלך התוכנית:



ניתן לראות שאכן האותות הופעלו כראוי, כך שפעולת ה- decoder התבצעה כמצופה. כעת נבחן שהתהליכים שהתבצעו בפעולות הם כמצופה.

עבור פקודת $ld\ r_3 \leftarrow M[2+r_0]$ שמתבצעת ב $cycles$ 5 נראה את התהליך הבא:

ה-cycle הראשון שבו מתבצע ה- fetch:

```
# *********Datapath Debug Section*****************
# time =        750000 ps
# Immidiate = UUUUUUUUUUUUUUUU
# A =         UUUUUUUUUUUUUUUU
# B =         0000000000000000
# C =         UUUUUUUUUUUUUUUU
# Cflag =     U
# Nflag =     U
# Zflag =     0
# OPC =       1111
# CregisterOut =   UUUUUUUUUUUUUUUU
# ***********************************************
# IRop =              1101
# Write Data to RF =  0000000000000000
# Read Data from RF = 0000000000000000
# ReadWriteAddressRF =          0000
# dataBus =           0000000000000000
# WriteAddressDataMem =     UUUUUU
# dataInDataMem =     0000000000000000
# **************** Status **********************
# Mem_wr =    0
# Cout =      0
# Cin =       0
# OPC =       1111
# Ain =       1
# RFin =      0
# RFout =     1
# RFaddr =    01
# IRin =      0
# PCin =      0
# PCsel =     01
# Imm1_in =   0
# Imm2_in =   0
# Mem_in =    0
# Mem_out =   0
#    Time: 750 ns  Iteration: 1  Instance: /datapth_tb/DataPathUnit
```

ניתן לראות שהתקבלה במחזור ה- fetch מספר הפקודה של load. כעת במחזור של ה- decode נרצה להתחיל את פעולת החיבור ב- ALU על מנת למצוא את הכתובת המתאימה של תא הזיכרון, כלומר במקרה שלנו את $r_0 + 2$. לשם כך, נכניס כבר בשלב ה- decode (כי אנחנו יודעים שאנחנו נמצאים ב– load) את הערך של הרגיסטר r_0 לרגיסטר A .

```
# ***************** Status ************************
# Mem_wr =     0
# Cout =       0
# Cin =        0
# OPC =        1111
# Ain =        1
# RFin =       0
# RFout =      1
# RFaddr =     01
# IRin =       0
# PCin =       0
# PCsel =      01
# Imm1_in =    0
# Imm2_in =    0
# Mem_in =     0
# Mem_out =    0
```

ניתן לראות כי Ain וגם RFout למעלה, כמצופה.
כעת במצב Itype_0 נתחיל בחישוב הכתובת של הזיכרון ולכן נרצה לחבר את רגיסטר A יחד עם הערך של ה- Immediate -

```
# ***************** Status ************************
# Mem_wr =     0
# Cout =       0
# Cin =        1
# OPC =        0000
# Ain =        0
# RFin =       0
# RFout =      0
# RFaddr =     10
# IRin =       0
# PCin =       0
# PCsel =      01
# Imm1_in =    0
# Imm2_in =    1
# Mem_in =     0
# Mem_out =    0
#    Time: 850 ns  Iteration: 1  Instance: /datapth_tb/DataPathUnit
# ** Note: ************************************************
# *********Datapath Debug Section*****************
# time =       950000 ps
# Immidiate =  0000000000000010
# A =          0000000000000000
# B =          0000000000000010
# C =          0000000000000010
```

ניתן לראות באדום שהערך של B הוא כערך ה- Immediate כמצופה ובצהוב ניתן לראות שהחישוב התבצע כמו שצריך ונשמר ברגיסטר C כתוצאה מהדגל Cin. כמו כן, ניתן לראות ש- OPC=0000 כלומר על מצב חיבור.
כעת במצב הבא Itype_1 , נרצה לקחת את מוצא רגיסטר C ולהכניס אותו בתור הכתובת ממנה נרצה לקרוא בזיכרון, כלומר לרגיסטר של קריאת הכתובת מה- DataMem במצב הבא Itype_2 והאחרון, נרצה להוציא את המידע מהזיכרון מהתא בערך שמצאנו ב ALU-ולשים אותו בתוך הרגיסטר –

```
# ************************************************
# IRop =                 1101
# Write Data to RF =     0000000000000010
# Read Data from RF =    UUUUUUUUUUUUUUUU
# ReadWriteAddressRF =           0011
# dataBus =              0000000000000010
# WriteAddressDataMem =      UUUUUU
# dataInDataMem =        0000000000000010
# ***************** Status ************************
# Mem_wr =     0
# Cout =       0
# Cin =        0
# OPC =        1111
# Ain =        0
# RFin =       1
# RFout =      0
# RFaddr =     10
# IRin =       0
# PCin =       1
# PCsel =      01
# Imm1_in =    0
# Imm2_in =    0
# Mem_in =     0
```

ניתן לראות שהערך שעל ה BUS-הוא הערך שנמצא בתא מספר 2 (הערך שמצאנו במוצא חישוב ה- ALU) הוא 2 והוא אכן נמצא על ה- BUS כמו כן, ערך זה הולך להיכנס ל- RF ברגיסטר המתאים כפי שניתן לראות בצהוב.

```
# ** Note: *************** Register File Content ***************
# R[0]  = 0000
# R[1]  = 0000
# R[2]  = 0000
# R[3]  = 0000
# R[4]  = 0000
# R[5]  = 0000
# R[6]  = 0000
# R[7]  = 0000
# R[8]  = 0000
# R[9]  = 0000
# R[10] = 0000
# R[11] = 0000
# R[12] = 0000
# R[13] = 0000
# R[14] = 0000
# R[15] = 0000
# ********************** UPDATED **********************
# R[0011] = 0002
# *****************************************************
#    Time: 1050 ns  Iteration: 1  Instance: /datapth_tb/DataPathUnit/RegFileModule
```

ניתן להמשיך ולעבור על כל הפקודות ב- TB ולראות שאכן ה- Datapath מתבצע כמצופה. נראה בקובץ זה רק את תוכן ה-Register על מנת לוודא תקינות ולא גם את התהליך של המחזורים הקודמים לו, אך וידאנו בעצמנו שאכן כל הערכים הם כמצופה.

לאחר פקודת ה– $Mov\ r_2 \leftarrow 5$

```
# ** Note: *************** Register File Content ***************
# R[0]  = 0000
# R[1]  = 0000
# R[2]  = 0000
# R[3]  = 0002
# R[4]  = 0000
# R[5]  = 0000
# R[6]  = 0000
# R[7]  = 0000
# R[8]  = 0000
# R[9]  = 0000
# R[10] = 0000
# R[11] = 0000
# R[12] = 0000
# R[13] = 0000
# R[14] = 0000
# R[15] = 0000
# ********************** UPDATED **********************
# R[0010] = 0005
# *****************************************************
#    Time: 1250 ns  Iteration: 1  Instance: /datapth_tb/DataPathUnit/RegFileModule
```

לאחר פקודת $Add\ r_4 \leftarrow r_3 + r_2$ נצפה שיהיה $r_4 \leftarrow 2 + 5 = 7$ ואכן-

```
# ** Note: *************** Register File Content ***************
# R[0]  = 0000
# R[1]  = 0000
# R[2]  = 0005
# R[3]  = 0002
# R[4]  = 0000
# R[5]  = 0000
# R[6]  = 0000
# R[7]  = 0000
# R[8]  = 0000
# R[9]  = 0000
# R[10] = 0000
# R[11] = 0000
# R[12] = 0000
# R[13] = 0000
# R[14] = 0000
# R[15] = 0000
# ********************** UPDATED **********************
# R[0100] = 0007
# *****************************************************
#    Time: 1650 ns  Iteration: 1  Instance: /datapth_tb/DataPathUnit/RegFileModule
```

לאחר פקודת $st\ M[1+r_0] \leftarrow r_4$ נצפה שיהיה $M[1] \leftarrow 7$ ואכן-

```
# ** Note: *************** Data Memory Content ***************
# 0  = 0000
# 1  = 0000
# 2  = 0002
# 3  = XXXX
# 4  = XXXX
# 5  = XXXX
# 6  = XXXX
# 7  = XXXX
# 8  = XXXX
# 9  = XXXX
# 10 = XXXX
# 11 = XXXX
# 12 = XXXX
# 13 = XXXX
# 14 = XXXX
# ********************** UPDATED **********************
# 01 = 0007
# ******************************************************
#    Time: 2150 ns  Iteration: 1  Instance: /datapth_tb/DataPathUnit/DataMemModule
```

ניתן לראות שאכן כל הפקודות ביצעו את התוכנית כפי שציפינו. כמו כן, בדקנו בצורה הזו את כל פקודות ה ISA-וכולן מתבצעות כנדרש.

סימולציה עבור המערכת כולה

בסימולציה הנוכחית, נרצה לאתחל את קבצי ה- txt המתאימים, כלומר את קובץ ה- Instruction וקובץ אתחול ה- DataMem. לאחר אתחול זה, נוכל להריץ סדרת פקודות בהתאם לקובץ הפקודות שהכנסנו, המותאמות ל -ISA שהגדרנו, ולבדוק האם הפקודות מתבצעות כראוי. אופן הבדיקה של ביצוע הפקודות הוא הן על ידי בחינת ההדפסות המתבצעות במהלך התוכנית והן על ידי בדיקת קובץ ה- txt של ה- DataMem במוצא התוכנית.
נאתחל את קבצי ה- txt בהתאם לתוכנית שהוצעה לנו בדף המשימה–

DTCMinit.txt

1	0014
2	000B
3	0002
4	0017
5	000E
6	0023
7	0006
8	0007
9	0030
10	0027
11	000A
12	000B
13	000C
14	000D
15	0000

ITCMinit.txt

1	D104
2	D205
3	C31F
4	C401
5	C50E
6	2113
7	2223
8	1621
9	8002
10	0640
11	7001
12	0600
13	E650
14	F000
15	0000
16	70FE

נזכיר כי המשימה אמורה לבצע את הפסאודו-קוד הבא–

```
int arr[14]={20,11,2,23,14,35,6,7,48,39,10,11,12,13}
int res;

void main(){

    R[1] = arr[4] & 31;
    R[2] = arr[5] & 31;

    if(R[2] >= R[1])
        res=0;
    else
        res=1;

    loop_forever;

}
```

כאשר ערך res = 1 והוא הערך שנצרב לזיכרון בעזרת פקודת store לתא מספר 0xE כתוצאה מפקודת E650 שמבצעת $st\ M[1 + r_5] \leftarrow r_6$
ניתן לראות שאכן בתא מספר 0xE התקבל הערך 1 שהגיע מהרגיסטר-R6

```
# ** Note: *************** Register File Content ***************
# R[0]  = 0000
# R[1]  = 000E
# R[2]  = 0003
# R[3]  = 001F
# R[4]  = 0001
# R[5]  = 000E
# R[6]  = FFF5
# R[7]  = 0000
# R[8]  = 0000
# R[9]  = 0000
# R[10] = 0000
# R[11] = 0000
# R[12] = 0000
# R[13] = 0000
# R[14] = 0000
# R[15] = 0000
# ********************** UPDATED **********************
# R[0110] = 0001
# *****************************************************
# ** Note: *************** Data Memory Content ***************
# 0  = 0014
# 1  = 000B
# 2  = 0002
# 3  = 0017
# 4  = 000E
# 5  = 0023
# 6  = 0006
# 7  = 0007
# 8  = 0030
# 9  = 0027
# 10 = 000A
# 11 = 000B
# 12 = 000C
# 13 = 000D
# 14 = 0000
# ********************** UPDATED **********************
# 0E = 0001
# *****************************************************
#    Time: 5750 ns  Iteration: 1  Instance: /top_tb/TopUnit/DataPathUnit/DataMemModule
```