Preparation Report LAB3

ADVANCED CPU ARCHITECTURE AND HARDWARE ACCELERATORS LAB

361.1.4693

Roy Kislev 206917064

Michael Grenader 208839845

מבוא

במעבדה זו רצינו לבנות תכנון בסיסי של CPU מסוג multi-cycle המכיל יחידת בקרה Control Unit יחד עם Datapath, כך שידע להריץ מספר הוראות בסיסיות המצורפות בISA הבאה –

Instruction Format	Decimal value	OPC	Instruction	Explanation	N	Z	C
R-Type	0	0000	add ra,rb,rc	R[ra]<=R[rb]+R[rc]	*	*	*
	1	0001	sub ra,rb,rc	R[ra]<=R[rb]-R[rc]	*	*	*
	2	0010	nop	R[0]<=R[0]+R[0]	-	-	-
	3	0011	*unused*				
J-Type	4	0100	jmp offset_addr	PC<=PC+1+offset_addr	-	-	-
	5	0101	jc /jhs offset_addr	If(Cflag==1) PC<=PC+1+offset_addr	-	-	-
	6	0110	jnc /jlo offset_addr	If(Cflag==0) PC<=PC+1+offset_addr	-	-	-
	7	0111	*unused*				
I-Type	8	1000	mov ra,imm	R[ra]<=imm	-	-	-
	9	1001	ld ra,imm(rb)	R[ra] <= M[imm+R[rb]]	-	-	-
	10	1010	st ra,imm(rb)	M[imm+R[rb]] <= R[ra]	-	-	-
	11	1011	done	Signals to TB that DTCM content is ready to be read	-	-	-

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

המימוש של יחידת הבקרה התבצע באמצעות FSM מסוג Mealy כפי שנלמד בקורס המקביל ויחידת הDatapath מתבצעת בצורה מקבילית כך שבהינתן הקווי בקרה שיוצאים מיחידת הבקרה, היחידה תבצע את המתבקש בcycle הנוכחי של הפקודה.

להלן המערכת שנרצה לממש –

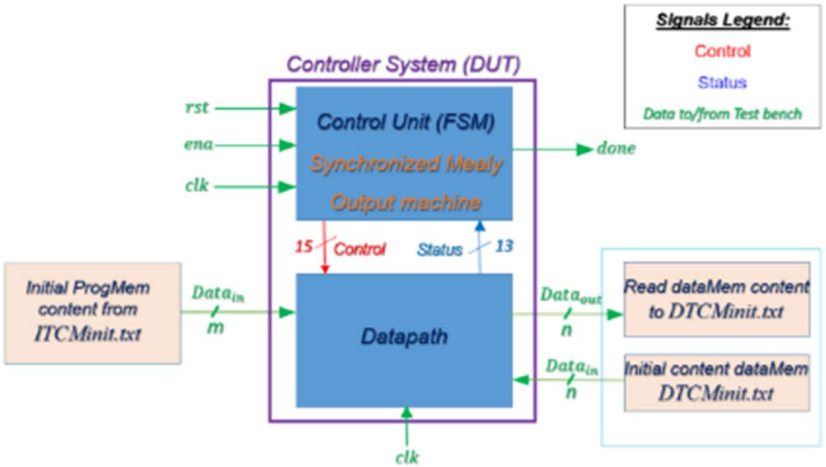


Figure 1: Overall DUT structure

המערכת שלנו תקבל 2 קבצי טקסט שישפיעו על פעילות הCPU יחד עם קווי בקרה שונים –

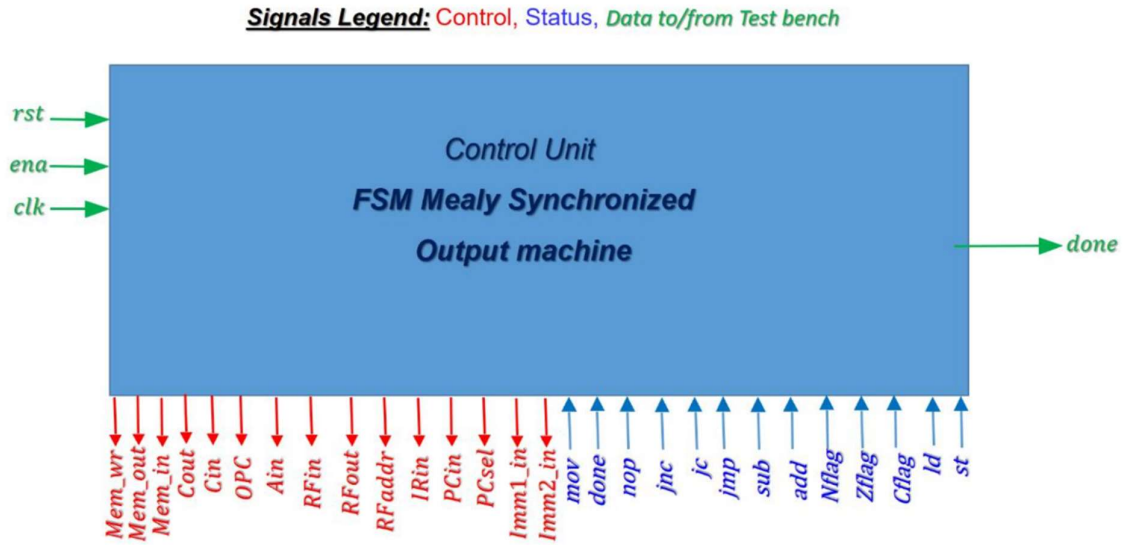
- ITCMinit.txt קובץ שיכיל את כל הInstructions של התוכנית אותה נרצה להריץ, כאשר הפקודות כתובת הHex. מאתחל את הProgram Memory
- DTCMinit.txt קובץ שיכיל את כל המידע שישב בתאי הזיכרון לפי הסדר – כלומר בתא 0 ישב הערך שיהיה בשורה הראשונה בקובץ וכך הלאה. מאתחל את הData Memory

- rst, clk, ena, done

בסיום הרצת התוכנית המערכת תוציא קובץ טקסט שמכיל את זיכרון הDataMem של המערכת בסיום התוכנית יחד עם דגל done.

Control Unit

דיאגרמת בלוק –

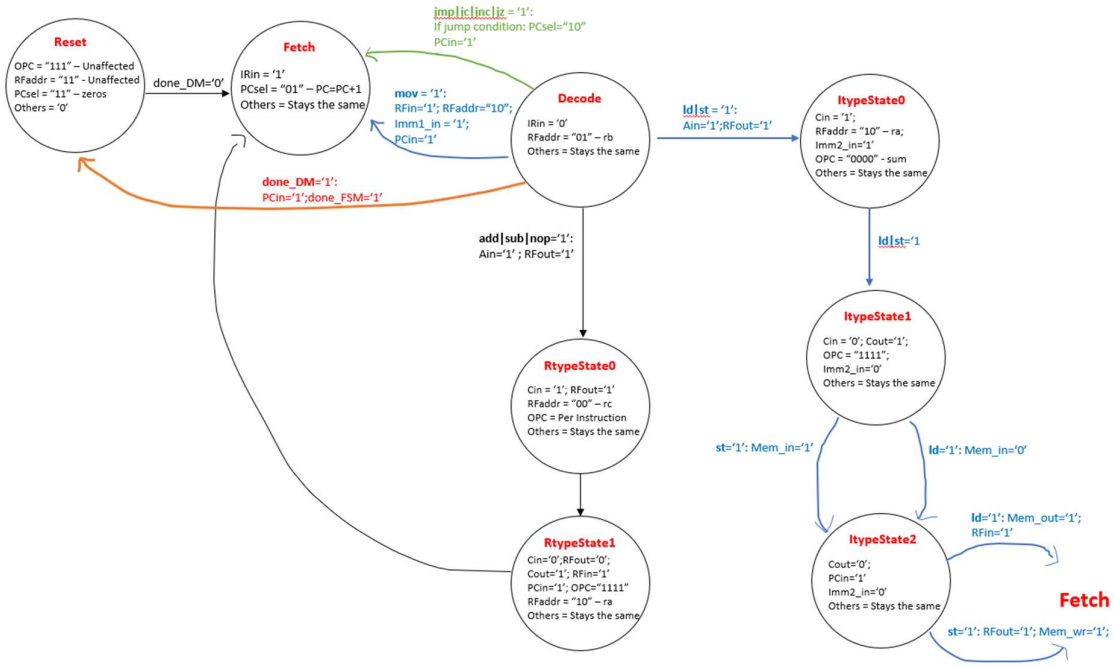


קווי הבקרה שהוא מוציא לעבר הDatapath הם בצבע אדום, קווי הStatus שהDatapath שולח לControl Unit שהם שולחים איזו פקודה מתבצעת כעת לאחר פענוח הפקודה מהIR הם בצבע כחול וקווי הTB הם בצבע ירוק.

תיאור המודול –

הControl Unit הוא החלק במערכת שניתן להקביל אותו למוח של המערכת, מטרתו היא לקבל את דגל הפקודה אותה נרצה לבצע שהיא נגזרת מיידית מהIR (רגיסטר ההוראה) ובהינתן הדגל הזה לעלות קווי בקרה מתאימים בהתאם לFSM המצורף בהמשך, כאשר קווי הבקרה האלו עוברים ליחידת הDatapath [ראו Datapath Unit] ומכתיבים לה את אופן פעילותה.

כתוצאה מיחידות הBUS והISA המצורפת על יחידת הבקרה לבצע את ההוראות שלה בכמה מחזורים ברציפות המתוארות בFSM הבא –



הFSM מכיל מספר סוגי פקודות בין היתר כתוצאה מהסוגים השונים של ההוראות המצורפות הISA (הוראות מסוג Rtype, Jtype, Itype). לשם כך חילקנו את הFSM הן למחזורים שקוראים אצל כולם Fetch, Decode ולמחזורים שמתפלגים כתלות בסוג הפקודה שמגיעה –

Fetch – חישוב כתובת הפקודה הבאה לביצוע, וקריאת הפקודה הבאה לביצוע מהזיכרון.

Decode – פיענוח הפקודה בDatapath ושליחתה ליחידת הבקרה שהתאם לכך יודעת איזו הוראה לבצע וכך את איזה דגלים לעלות. כמו כך, הבאת ערכי הרגיסטרים בהם הפקודה משתמשת בחלק מהפקודות.

Rtype – פקודות המכילות את המבנה הבא $OPC|Ra|Rb|Rc$.

Jtype – פקודות המכילות את המבנה הבא $OPC|0000000|offset$.

Itype – פקודות המכילות את המבנה הבא $OPC|Ra|imm$ או $OPC|Ra|Rb|imm$.

Datapath Unit

דיאגרמת בלוק –

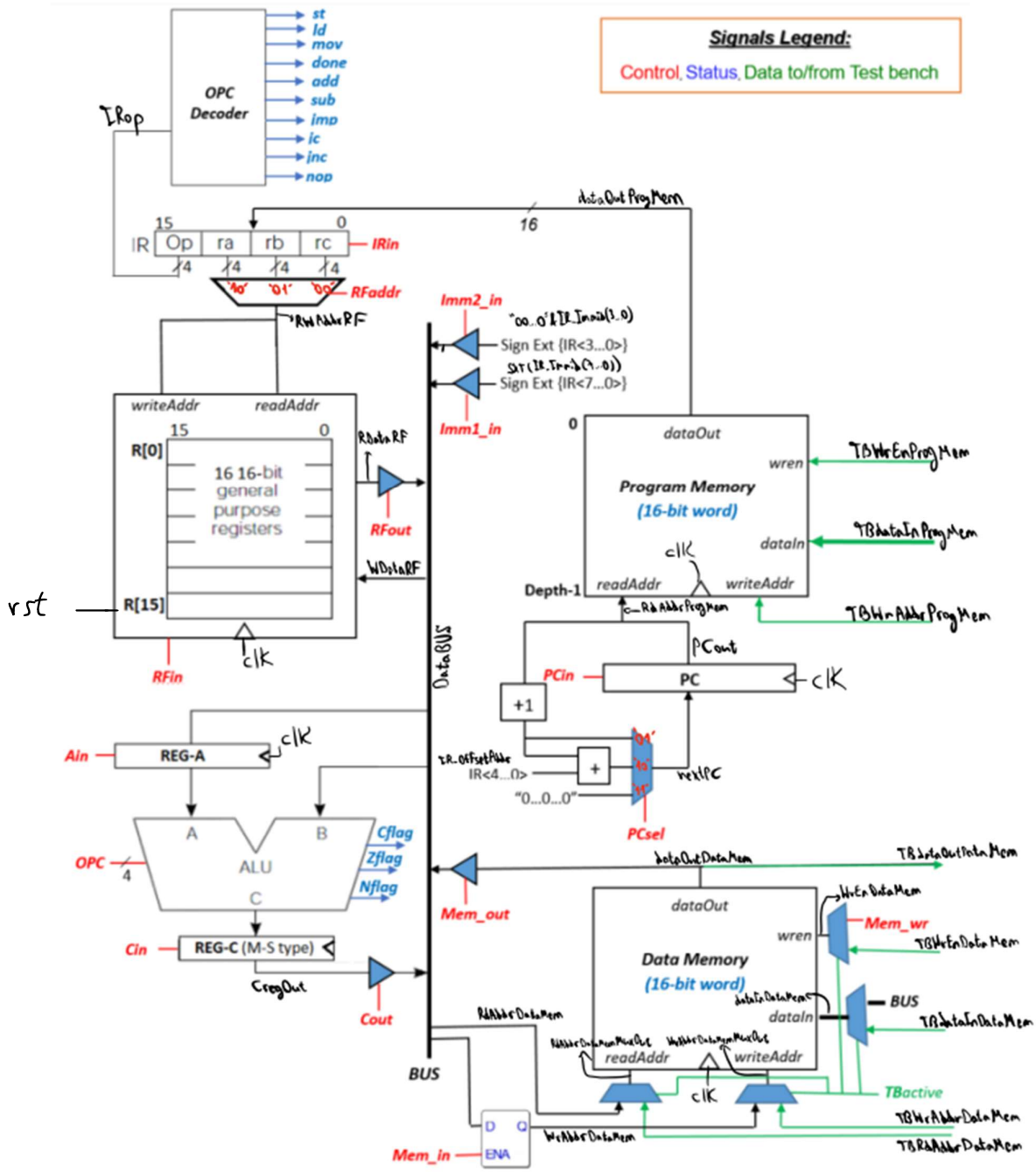


Figure 2: Datapath structure

תיאור המודול –

הDatapath Unit הוא החלק במערכת שניתן להקביל אותו לשרירים של המערכת, מטרתו היא לקבל את הפקודה אותה נרצה לבצע מהprogMem וממנה לחלץ באמצעות decoder את דגל הסטטוס של הפקודה אותה נרצה לבצע ולהעבירה ליחידת הבקרה. כמו כן, מטרתה העיקרית של יחידה זו היא בהינתן קווי הבקרה לבצע את המודולים הסינכרוניים והא-סינכרוניים שיביאו לפעילות הפקודה הנדרשת.

נראה דוגמה של אחת הפקודות, נניח נרצה לבצע את הפקודה הבאה: 9202. כאשר נקרא את הנתון הנ"ל אל הProgMem, נקבל את הקידוד הבא בבינארי:

1001001000000010

ומכאן נוכל לחלץ את הפורמט של ההוראה הדרושה:

$$ld\ r2, imm(r0): R[r2] <= M\big[2 + R[r0]\big]$$

בשלב ה-fetch נרצה להביא את הפקודה הדרושה אל תוך רגיסטר ההוראה, IR. מכאן גם יעלה דגל הסטטוס המתאים של איזו פקודה נרצה לבצע, במקרה כאן ld='1'.

בשלב ה-decode, נרצה להוציא לבאס את ערך רגיסטר rb שבמקרה שלנו יהיה 0 (r0 מאותחל להיות 0 בהתחלה ונשים 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), לכן נעלה את RFaddr="01" ו-RFout='1' כדי להוציא את הכתובת המתאימה לRF ואז להוציא את הערך של R[rb] לבאס. כדי לחסוך מחזורי שעון, בשלב זה גם נבצע Ain='1' כדי שבמחזור הבא A יקבל את הערך שבבאס. (REG-A קורה בעליית שעון ומתעדכן בסוף הprocess)

בשלב ItypeState0, נרצה לבצע את החיבור של A עם ערך Imm כדי למצוא את הכתובת שנרצה לקרוא מזיכרון הData. נוציא לשם כך את Imm2 לבאס עם Imm2_in='1', ונתן לALU פקודת חיבור עם OPC="0000", ובשלב זה תוצאת החיבור תהיה Imm+R[rb]. נרצה שבמחזור הבא מוצא REG-C יחזיק את תוצאת החיבור, לכן נתן לו Cin='1'. בנוסף גם נשנה את כתובת הרגיסטר של RF שנרצה לכתוב אליו לra כלומר RFaddr="10".

בשלב ItypeState1, נרצה להוציא את תוצאת החיבור לכתובת שנרצה לקרוא ממנה בDataMem, לכן נוציא את התוצאה (מוצא REG-C) לבאס עם Cout='1' (לא לשכוח לוודא ש-Imm2_in=0 כדי שלא תהיה התנגשות בבאס)

בשלב ItypeState2, נרצה להוציא את הערך בזיכרון בכתובת שחישבנו (Imm+R[rb]), לכן נוציא את המידע החוצה לבאס עם Mem_out='1' (לא לשכוח Cout='0' כדי למנוע התנגשות), נרצה לכתוב לרגיסטר R[ra], לכן נתן אפשור כתיבה עם RFin='1' ונוודא ש-RFaddr="10" כדי לכתוב ל-ra. נקדם את PC עם PCin='1' ונחזור לשלב ה-Fetch

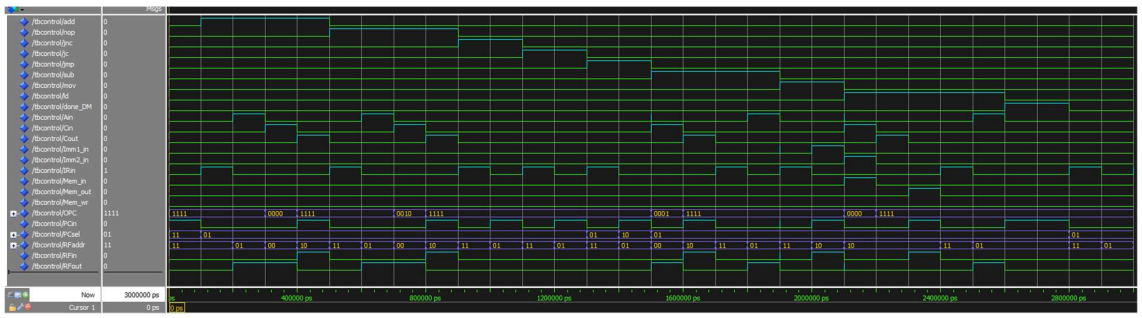
תוצאות סימולציה

נחלק את תוצאות הסימולציה לשלושה:

- סימולציה עבור ה-Control Unit
- סימולציה עבור ה-Datapath Unit
- סימולציה עבור המערכת כולה

סימולציה עבור ה-Control Unit

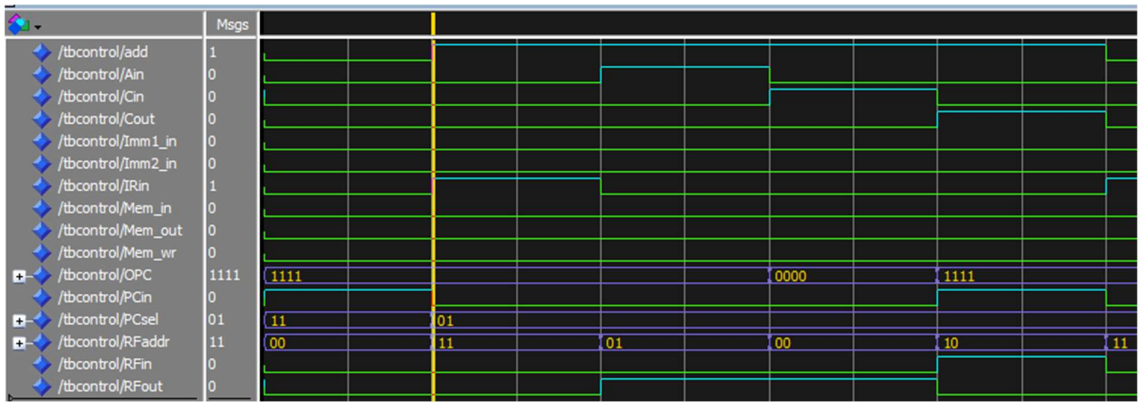
במהלך סימולציה זו נעלה דגלים של פקודות שונות שהDatapath אמור להעביר לControl Unit לאחר ביצוע הdecode ואז נבדוק שקווי הבקרה שעולים בכל אחד מהמחזורים הם קווי הבקרה שאנחנו מצפים שיהיו. עבור כל פקודה, נעלה את הדגל לאורך כמות הcycles שלוקח לה. להלן תוצאת כל הסימולציה כך שסדר הפקודות שארכו הן add → nop → jnc → jc → jmp → sub → mov → ld → done_DM.



כעת נסקור את כל הפקודות בנפרד ונוודא שהדגלים שעולים הם אכן כמצופה לפי הFSM המצורף.

פקודת add –

בפקודת add נצפה שנהיה במצבי הFSM הבאים fetch → decode → Rtype_0 → Rtype_1. להלן סימולציית הגלים –

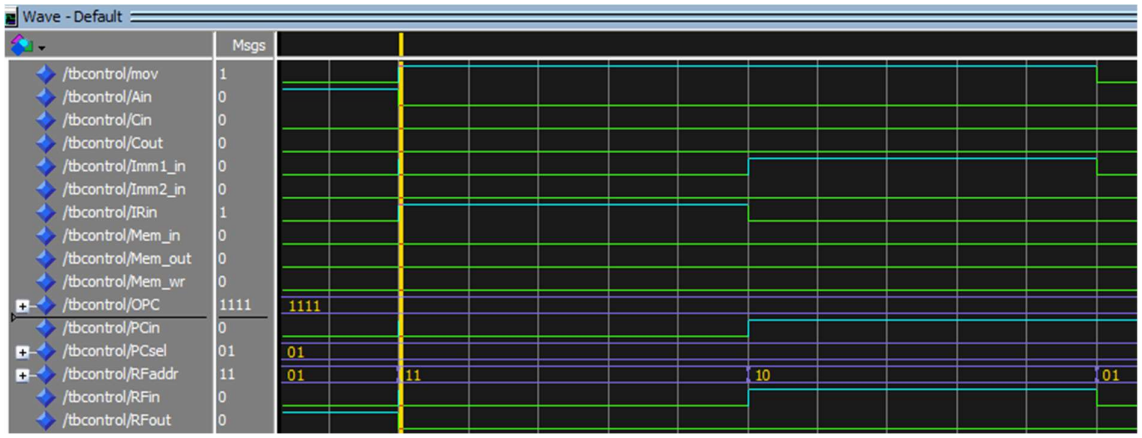


ניתן לראות שפקודת הadd מכילה *4 cycles* כאשר בcycle הראשון *IRin* עולה כתוצאה מהגדרת הfetch במהלכה נרצה להכניס לIR את ערך הפקודה לביצוע ולאחר מכין מתחיל שלב הdecode במהלכו נרצה להחזיר את הדגל המתאים אבל נרצה לנצל את הBUS גם בcycle הזה לכן נשים רגיסטר על הBUS כדי שיכנס לרגיסטר A כחלק מהגדרת פעולת add. לאחר מכן, נשים על הBUS את הרגיסטר השני כדי שיכנס

לכניסת B של הALU ויתבצע החישוב OPC="0000" שמסמן חישוב. כמו כן במחזור זה נרצה שיעלה דגל Cin כדי שתוצאת החישוב תכנס לרגיסטר C ובמחזור הבא והאחרון נרצה להוציא את תוצאת החישוב שמוחזקת בC לתוך הBUS כדי שיכנס לרגיסטר דרך הRF ולכן נעלה את דגל Cout.

פקודת mov

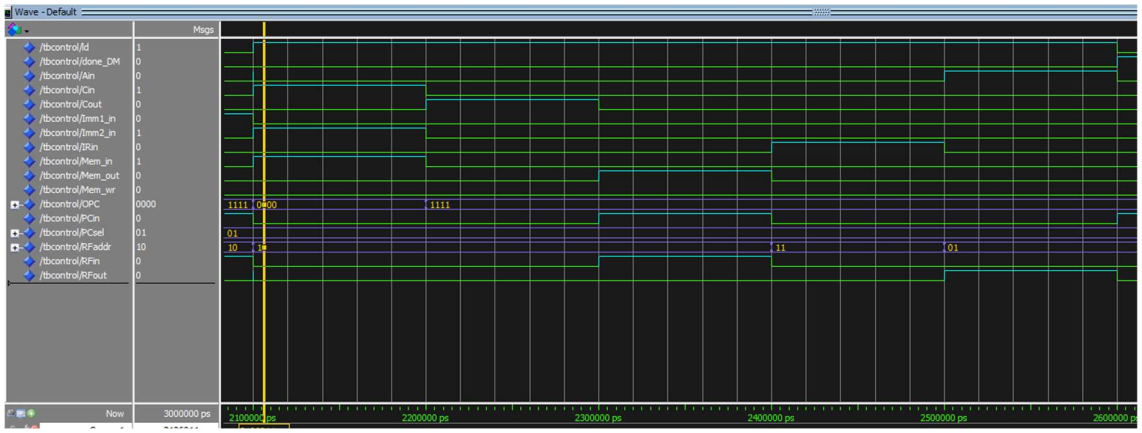
בפקודת mov נצפה שנהיה במצבי הFSM הבאים fetch → decode. להלן סימולציית הגלים –



ניתן לראות שפקודת הmov מכילה $cycles$ 2 כאשר בcycle הראשון $IRin$ עולה כתוצאה מהגדרת הfetch במהלכה נרצה להכניס לIR את ערך הפקודה לביצוע ולאחר מכין מתחיל שלב הdecode במהלכו נרצה להחזיר את הדגל המתאים אבל נרצה לנצל את הBUS כך שנוכל כבר בשלב זה לבדוק שאם דגל הפקודה שעלה הוא mov אז נעביר את הערך מהמספר שנכנס דרך הImmediate לעבר הRF ולכן בcycle הזה לכן נשים נעלה את Imm_1 כדי להכניס את המספר שנכנס לImmediate לBUS וכמו כן נעלה את RFin על מנת להכניס להכניס את המידע מהBUS לעבר הרגיסטר המתאים ששמנו בRFaddr.

פקודת ld

בפקודת ld נצפה שנהיה במצבי הFSM הבאים fetch → decode → Itype_0 → Itype_1 → Itype_2. להלן סימולציית הגלים –



פקודה זו מבצעת $ld\ ra, imm(rb): R[ra] <= M[imm + R[rb]]$. בשלב ה-fetch נרצה להביא את הפקודה הדרושה אל תוך רגיסטר ההוראה, IR. מכאן גם יעלה דגל הסטטוס המתאים של איזו פקודה נרצה לבצע, במקרה כאן ld='1'.

בשלב ה-decode, נרצה להוציא לבאס את ערך רגיסטר rb שבמקרה שלנו יהיה 0 (r0 מאותחל להיות 0 בהתחלה ונשים 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), לכן נעלה את RFaddr="01" ו-RFout='1' כדי להוציא את הכתובת המתאימה לRF ואז להוציא את הערך של R[rb] לבאס. כדי לחסוך מחזורי שעון, בשלב זה גם נבצע Ain='1' כדי שבמחזור הבא A יקבל את הערך שבבאס. (REG-A קורה בעליית שעון ומתעדכן בסוף הprocess)

בשלב ItypeState0, נרצה לבצע את החיבור של A עם ערך Imm כדי למצוא את הכתובת שנרצה לקרוא מזיכרון הData. נוציא לשם כך את Imm2 לבאס עם Imm2_in='1', ונתן לALU פקודת חיבור עם OPC="0000", ובשלב זה תוצאת החיבור תהיה Imm+R[rb]. נרצה שבמחזור הבא מוצא REG-C יחזיק את תוצאת החיבור, לכן נתן לו Cin='1'. בנוסף גם נשנה את כתובת הרגיסטר של RF שנרצה לכתוב אליו לra כלומר RFaddr="10".

בשלב ItypeState1, נרצה להוציא את תוצאת החיבור לכתובת שנרצה לקרוא ממנה בDataMem, לכן נוציא את התוצאה (מוצא REG-C) לבאס עם Cout='1' (לא לשכוח לוודא ש-Imm2_in=0 כדי שלא תהיה התנגשות בבאס)

בשלב ItypeState2, נרצה להוציא את הערך בזיכרון בכתובת שחישבנו (Imm+R[rb]), לכן נוציא את המידע החוצה לבאס עם Mem_out='1' (לא לשכוח Cout='0' כדי למנוע התנגשות), נרצה לכתוב לרגיסטר R[ra], לכן נתן אפשור כתיבה עם RFin='1' ונוודא ש-RFaddr="10" כדי לכתוב ל-ra. נקדם את PC עם PCin='1' ונחזור לשלב ה-Fetch.

סימולציה עבור ה-Datapath Unit

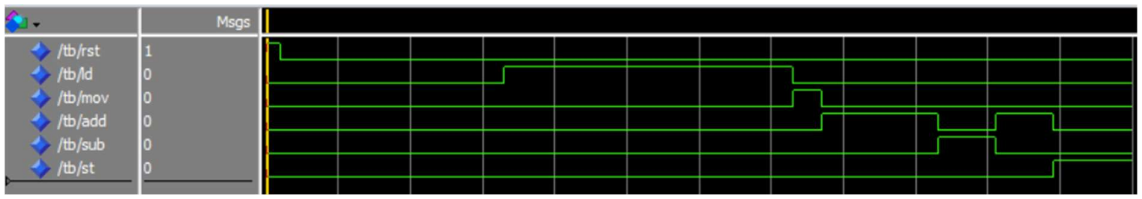
עבור הסימולציה הנ"ל, השתמשנו בreport בdataPath עצמו כדי להדפיס את הסיגנלים וקווי הבקרה המתאימים כדי לדבג בצורה יעילה יותר, כך שנוכל לראות גם את ערכי הסיגנלים וגם את הזיכרונות – הן את הDataMem והן את הRegFile. בסימולציה זו נעלה את הדגלים המתאימים לפי הFSM של קווי הבקרה שאמורים להיכנס לDatapath ונבדוק האם התקבלו ערכים תקינים כפי שהיינו מצפים מהפעולה לבצע.

בTB זה נבצע את הפעולות הבאות

$$Load\ r2 \leftarrow M[2+r0], Load\ r4 \leftarrow M[4+r0], Load\ r9 \leftarrow M[9+r0], Load\ r13 \leftarrow M[13+r0], Mov\ r1 \leftarrow 1$$

$$Add\ r2 \leftarrow r2+r13, Add\ r3 \leftarrow r4+r9, Sub\ r6 \leftarrow r2-r3, Add\ r6 \leftarrow r1+r0, Store\ M[15+0] \leftarrow r6$$

להלן הדגלים שעלו בתוכנית זו –



ניתן לראות שאכן הדגלים התבצעו כראוי, כך שפעולת ה*decoder* התבצע כמצופה. כעת נראה שהתהליכים שהתבצעו בפעולות הם כמצופה.

עבור פקודת $Ld\ r2 \leftarrow M[2+r0]$ שמתבצעת ב*cycles* 5 מתקיים. ה*cycle* הראשון שבמהלכו מתבצע ה*fetch* –

```
# ************************************************
# IRop =                1001
# neg =                 0
# Write Data to RF =    0000000000000000
# Read Data from RF =   0000000000000000
# RWAddrRF =            0000
# dataBus =             0000000000000000
# WrAddrDataMem =       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
```

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 $2+r0$. לשם כך, נכניס כבר בשלב ה*decode* (כי אנחנו יודעים שאנחנו נמצאים ב*load*) את הערך של הרגיסטר $r0$ 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
```

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: 1950 ns  Iteration: 1
# ** Note: **********Datapath Deb
# time =       2050000 ps
# Immidiate =  0000000000000010
# A =          0000000000000000
# B =          0000000000000010
# C =          0000000000000010
```

ניתן לראות בירוק 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 =                1001
# neg =                 0
# Write Data to RF =    0000000000000010
# Read Data from RF =   UUUUUUUUUUUUUUUU
# RWAddrRF =            0010
# dataBus =             0000000000000010
# WrAddrDataMem =       000010
# 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
```

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. כמו כן, ניתן לראות שאכן רגיסטר מספר 2 התעדכן להכיל את הערך 2 –

```
# ** Note: *************** Register File Content *******
# R[0]  = 0000
# R[1]  = XXXX
# R[2]  = XXXX
# R[3]  = XXXX
# R[4]  = XXXX
# R[5]  = XXXX
# R[6]  = XXXX
# R[7]  = XXXX
# R[8]  = XXXX
# R[9]  = XXXX
# R[10] = XXXX
# R[11] = XXXX
# R[12] = XXXX
# R[13] = XXXX
# R[14] = XXXX
# R[15] = XXXX
# ********************** UPDATED *********************
# R[0010] = 0002
# ************************************************
```

כעת התהליך התבצע עבור מספר רגיסטרים, ניתן לראות פה לאחר כל פקודות ה*load* שזה תוכן ה*Register File* –

```
# ** Note: ***************** Register File Content **
# R[0]  = 0000
# R[1]  = XXXX
# R[2]  = 0002
# R[3]  = XXXX
# R[4]  = 0004
# R[5]  = XXXX
# R[6]  = XXXX
# R[7]  = XXXX
# R[8]  = XXXX
# R[9]  = 0009
# R[10] = XXXX
# R[11] = XXXX
# R[12] = XXXX
# R[13] = XXXX
# R[14] = XXXX
# R[15] = XXXX
# ************************ UPDATED ********************
# R[1101] = 000D
# ************************************************
```

ניתן להמשיך ולעבור על כל הפקודות ב*TB* ולראות שאכן ה*datapath* מתבצע כמצופה. נראה בקובץ זה רק את תוכן ה*RegFile* על מנת לוודא תקינות ולא גם את התהליך של המחזורים הקודמים לו, אך וידאנו בעצמנו שאכן כל הערכים הם כמצופה. לאחר פקודת ה*mov* –

```
# ** Note: ***************** Register File Content
# R[0]  = 0000
# R[1]  = XXXX
# R[2]  = 0002
# R[3]  = XXXX
# R[4]  = 0004
# R[5]  = XXXX
# R[6]  = XXXX
# R[7]  = XXXX
# R[8]  = XXXX
# R[9]  = 0009
# R[10] = XXXX
# R[11] = XXXX
# R[12] = XXXX
# R[13] = 000D
# R[14] = XXXX
# R[15] = XXXX
# ************************ UPDATED ******************
# R[0001] = 0001
# ************************************************
```

לאחר פקודת $add\ r2 \leftarrow r2 + r13$ נצפה שיהיה $r2 \leftarrow 2 + 13 = 15 = 0xF$, ואכן –

```
** Note: ***************** Register File Content
R[0]  = 0000
R[1]  = 0001
R[2]  = 0002
R[3]  = XXXX
R[4]  = 0004
R[5]  = XXXX
R[6]  = XXXX
R[7]  = XXXX
R[8]  = XXXX
R[9]  = 0009
R[10] = XXXX
R[11] = XXXX
R[12] = XXXX
R[13] = 000D
R[14] = XXXX
R[15] = XXXX
************************ UPDATED ******************
R[0010] = 000F
************************************************
```

לאחר פקודת $add\ r3 \leftarrow r4 + r9$ נצפה שיהיה $r3 \leftarrow 4 + 9 = 13 = 0xD$, ואכן –

```
** Note: ***************** Register File Content
R[0]  = 0000
R[1]  = 0001
R[2]  = 000F
R[3]  = XXXX
R[4]  = 0004
R[5]  = XXXX
R[6]  = XXXX
R[7]  = XXXX
R[8]  = XXXX
R[9]  = 0009
R[10] = XXXX
R[11] = XXXX
R[12] = XXXX
R[13] = 000D
R[14] = XXXX
R[15] = XXXX
************************ UPDATED ******************
R[0011] = 000D
************************************************
```

לאחר פקודת $sub\ r6 \leftarrow r2 - r3$ נצפה שיהיה $r6 \leftarrow 15 - 13 = 2$, ואכן –

```
** Note: *************** Register File Content
R[0]  = 0000
R[1]  = 0001
R[2]  = 000F
R[3]  = 000D
R[4]  = 0004
R[5]  = XXXX
R[6]  = XXXX
R[7]  = XXXX
R[8]  = XXXX
R[9]  = 0009
R[10] = XXXX
R[11] = XXXX
R[12] = XXXX
R[13] = 000D
R[14] = XXXX
R[15] = XXXX
********************** UPDATED ****************
R[0110] = 0002
***********************************************
```

לאחר פקודת $add\ r6 \leftarrow r1 + r0$ נצפה שיהיה $r6 \leftarrow 1 + 0 = 1$, ואכן –

```
** Note: *************** Register File Content
R[0]  = 0000
R[1]  = 0001
R[2]  = 000F
R[3]  = 000D
R[4]  = 0004
R[5]  = XXXX
R[6]  = 0002
R[7]  = XXXX
R[8]  = XXXX
R[9]  = 0009
R[10] = XXXX
R[11] = XXXX
R[12] = XXXX
R[13] = 000D
R[14] = XXXX
R[15] = XXXX
********************** UPDATED ****************
R[0110] = 0001
***********************************************
```

לאחר פקודת $store\ M[15 + 0] \leftarrow r6$ נצפה שיהיה $M[15] \leftarrow 1$, ואכן –

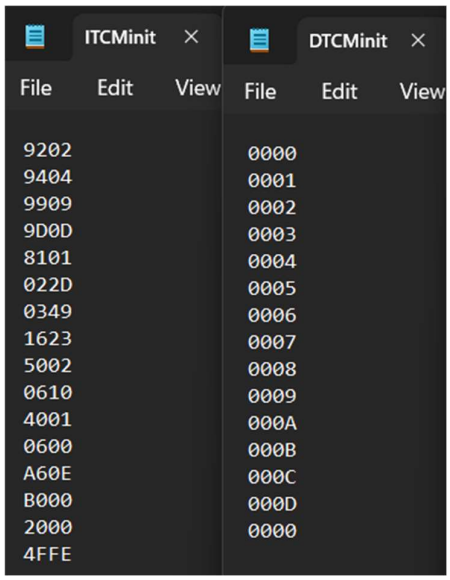
```
** Note: *************** Data Memory Content
0  = 0000
1  = 0001
2  = 0002
3  = 0003
4  = 0004
5  = 0005
6  = 0006
7  = 0007
8  = 0008
9  = 0009
10 = 000A
11 = 000B
12 = 000C
13 = 000D
14 = 0000
********************** UPDATED ***************
0E = 0001
***********************************************
```

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

סימולציה עבור המערכת כולה

בסימולציה זו נרצה לאתחל את קבצי הtxt המתאימים, כלומר את קובץ הInstruction וקובץ אתחול הDataMem. לאחר אתחול זה, נוכל להריץ סט פקודות בהתאם לקובץ הפקודות שהכנסו שמותאמות לISA שהכנסנו ולבדוק האם הפקודות מתבצעות כראוי. אופן הבדיקה האם הבדיקות התבצעו כראוי הוא הן על ידי מבט על ההדפסות שמתבצעות במהלך התוכנית והן על ידי בדיקת קובץ הtxt של הDataMem במוצא התוכנית.

נאתחל את קבצי הtxt בהתאם לתוכנית שהוצעה לנו בדף המשימה –



נזכור כי המשיה אמורה לבצע את הפסאודו קוד הבא –

```
int arr[14]={0,1,2,3,4,5,6,7,8,9,10,11,12,13}
int res=0;

void main(){

    R[3] = arr[2]+arr[13];
    R[2] = arr[4]+arr[9];

    if(R[2] >= R[3])
        res=0;
    else
        res=1;

    loop_forever;

}
```

כאשר ערך $res = 1$ והוא הערך שנצרב לזיכרון בעזרת פקודת store לתא מספר $0xE$, כתוצאה מפקודת $A60E$ שמבצעת $st\ M[E + r0] = r6$.

ניתן לראות שאכן בתא מספר $0xE$ התקבל הערך 1 שהגיע מהרגיסטר r6 –