מבוא:

במעבדה זאת התבקשנו לבנות CPU בסיסי מסוג multi-cycle המורכבת מ-2 תת מערכות: control unit ו datapath.

3. Controller-based system:

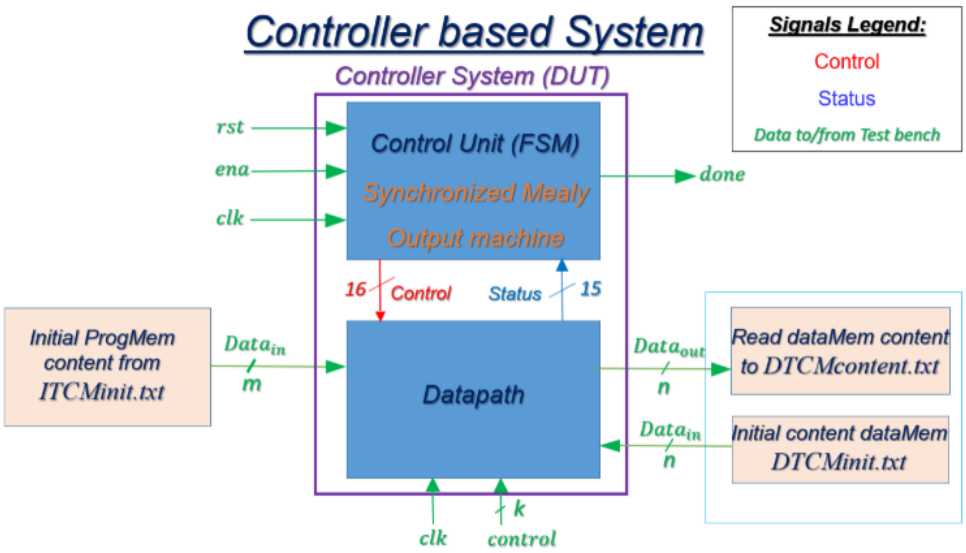


Figure 1: Overall DUT structure

המימוש של יחידת ה control מיושם על ידי מכונת מצבים סופית FMS מסוג MEALY, ויחידת ה datapath מיושמת בצורה מקבילית.

<u>המקשר בין 2 היחידות המרכזיות הינם סיגנלים מ2 סוגים:</u>

- סיגל בקרה- סיגנל היוצא מ control אל datapath
- סיגנל מצב- סיגנלים היוצאים מdatapath אל הcontrol

המערכת הזאת הינה מערכת שבהקבלה לגוף האנושי מתפקדת כמוח- יחידת הבקרה, וגוף- יחידת מעבר המידע (אחראית על ביצוע הפעולות).

<u>הסיגנלים הירוקים הינם סיגנלי input למערכת המתקבלים על ידי קבצי טקסט באופן הבא:</u>

- ITCMNIMIT - קובץ בינארי המכיל את הפקודות של התוכנית אותה נרצה להריץ, ומבצע אתחול ל PM.
- DTCMINIT – קובץ בינארי המכיל את במידע היושב בזיכרון, ומאתחל את ה DM.

- DTCMCOTANT - בסיום הרצת התוכנית המבוקשת נקבל קובץ טקסט בינארי המכיל את זיכרון הDM עם חיווי done מתאים.

בנוסף, המעבד הנ"ל נדרש לבצע את הפעולות הלוגיות והאריתמטיות הבאות:

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	-	-	-

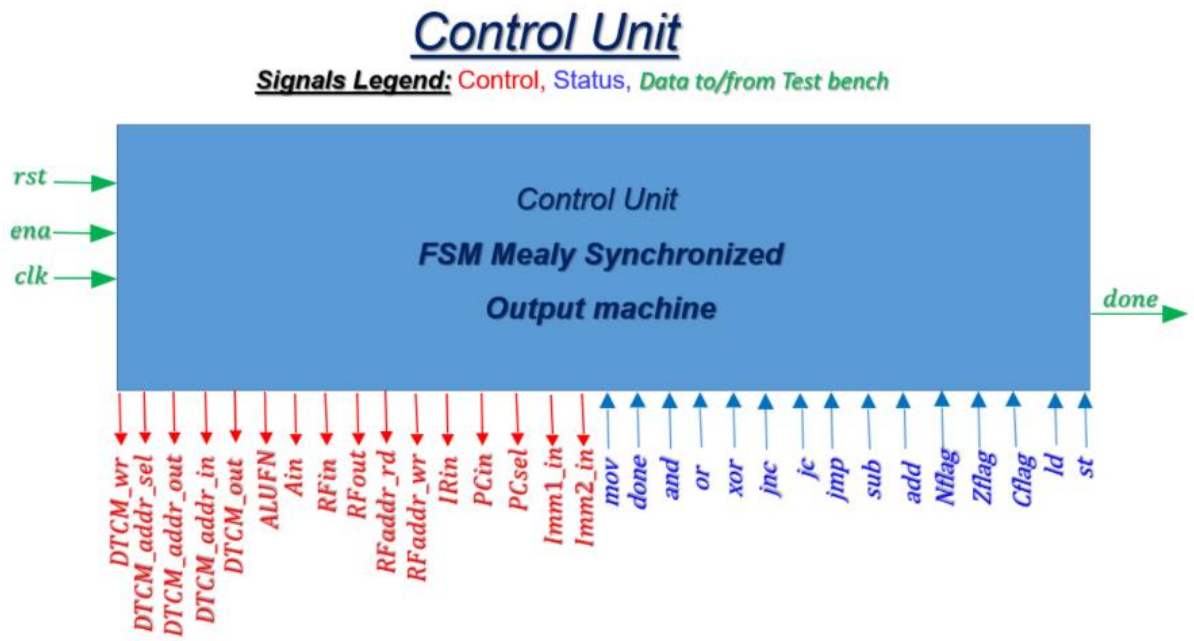
Note: * The status flag bit is affected , - The status flag bit is not affected

בפורמט הבא:

ADDRESS MODE	SYNTAX	INSTRUCTION FORMAT	EXAMPLE	OPERATION
Register	add ra,rb,rc	R-Type	add r4,r2,r1	$R[4] \leftarrow R[2] + R[1]$
Immediate	mov ra,imm	I-Type	mov r5,0x30 mov r5,Var	$R[5] \leftarrow 0x30$ $R[5] \leftarrow Var$
Direct	ld ra,imm(r0)		ld r4,0x20(r0) ld r4, Var(r0)	$R[4] \leftarrow M[0x20]$ $R[4] \leftarrow M[Var]$
Indirect	ld ra,0(rb)		ld r4,0(r3)	$R[4] \leftarrow M[R[3]]$
Indexed	ld ra,imm(rb)		ld r4,0x20(r3) ld r4, Var(r3)	$R[4] \leftarrow M[0x20 + R[3]]$ $R[4] \leftarrow M[Var + R[3]]$

Table 1: SRMC-I Addressing Mode

:CONTROL UNIT



- INPUT: סיגנלי מצב שמתקבלים מה DATAPATH.
- OUTPUT: סיגנלי בקרה שנכנסים ל DATAPATH.

כך בעצם המערכת יודעת לבצע את הפקודה הרצויה.

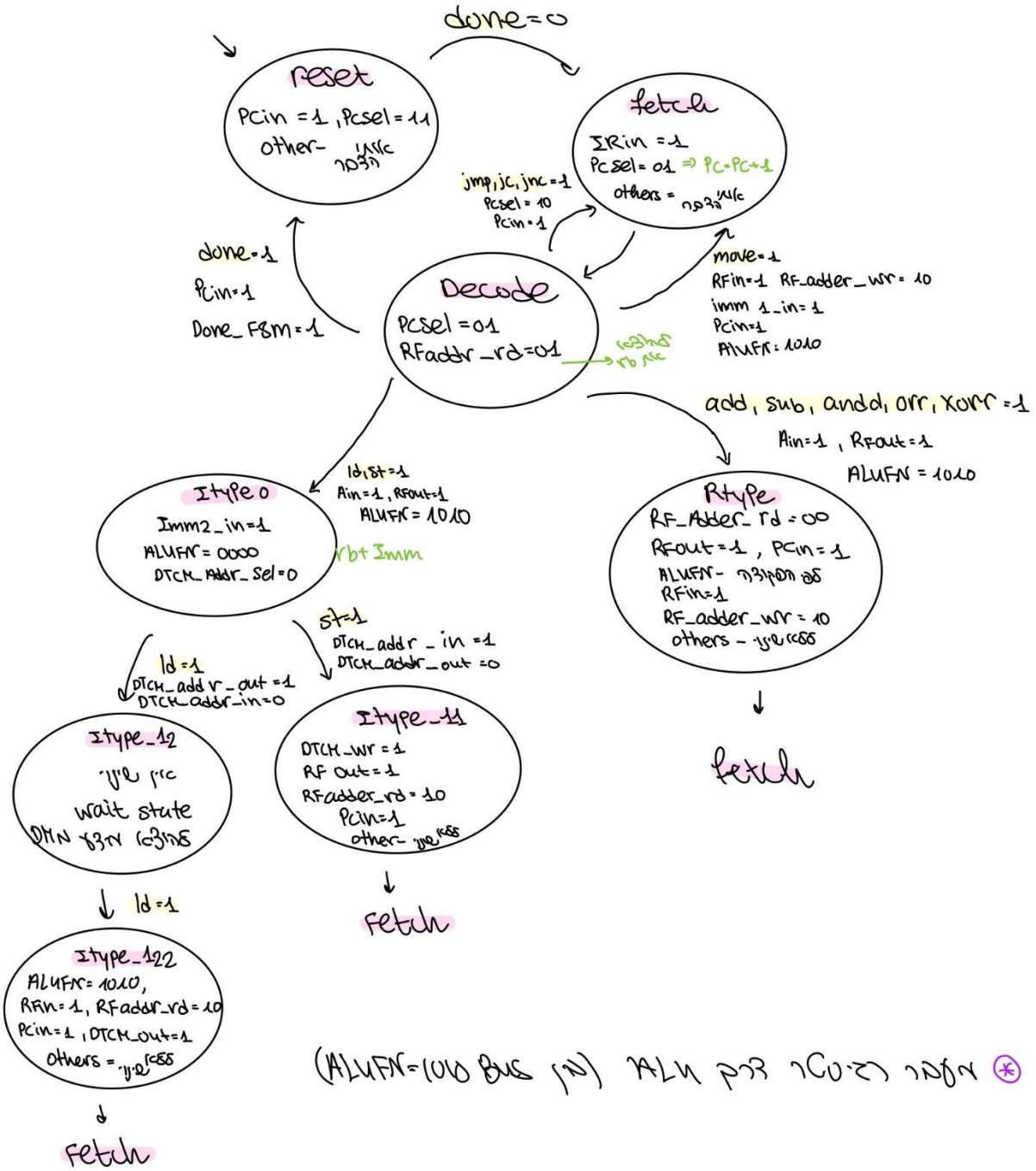
מערכת הבקרה משמשת המוח של המערכת, היא מערכת FSM כך שתהליך קבלת פקודה עובר בין מצבים אותם נתאר בדיאגרמת מצבים.

בעת כניסת פקודה למערכת, המערכת מפענחת את הפקודה הרצויה על ידי רכיב ה IR. בהינתן הדגל המתאים, היא מעלה את הדגלים הרלוונטיים לביצוע שלבי הפקודה בקווי הבקרה והמערכת תיישם את הפקודה המבוקשת.

מערכת המצבים שתכננו מיישמת את הפקודות הרלוונטיות, ומתוארת על ידי מצבים שונים, מכלליים יותר כמו FETCH,DECODE לכלליים פחות עקב חלוקת הפקודות ל 3 סוגים עיקריים RYPE, JTYPE, ITYPE.

המערכת מחולקת CYCLES לפי הפעולה המתבקשת ומעבר בין מצבים מהווה הדלקת דגלים שונים במערכת.

FSM:



* אפשר לבטל את הצורך בALU (אין BUS ALUFN=1010)

מצב פירוט מחדש:

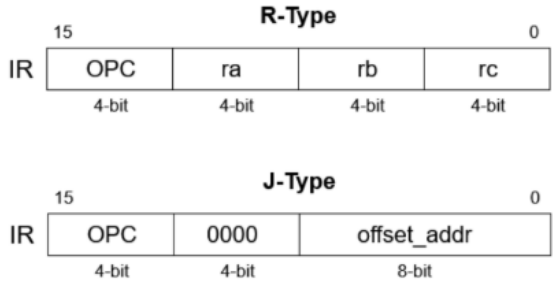
ALUFN ⇐ 1011
Ain ⇐ 0
RFin ⇐ 0
RFout ⇐ 0
RFadder_wr ⇐ 11
RFadder_rd ⇐ 11
IRin ⇐ 0
Pcin ⇐ 0
PCsel ⇐ 01
Imm1_in ⇐ 0
Imm2_in ⇐ 0
DTCM_wr ⇐ 0
DTCM_out ⇐ 0
DTCM_addr_sel ⇐ 0
DTCM_addr_in ⇐ 0
DTCM_addr_out ⇐ 0
Done_FSM ⇐ 0

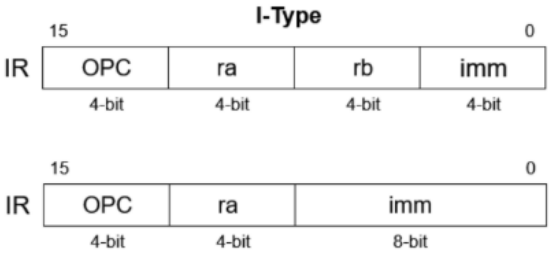
שאת הרגיסטרים מאתחלת:

ra = 10
rb = 01
rc = 00

המצבים השונים:

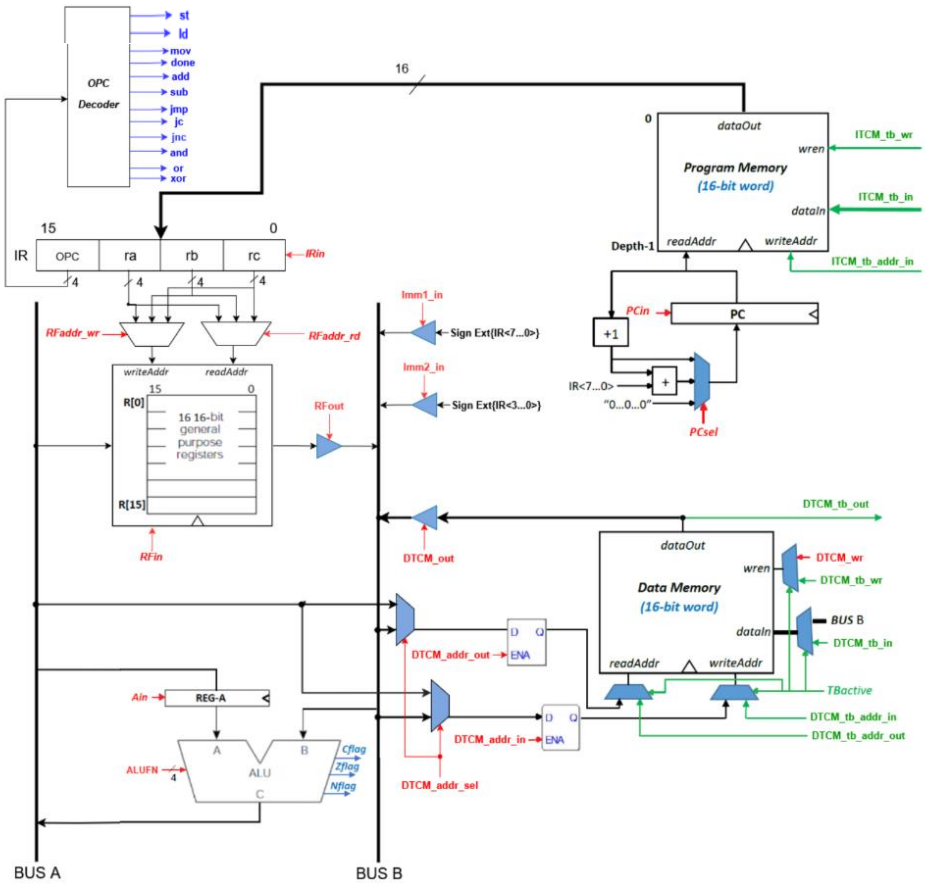
- RST – מצב התחלת המערכת.
- FETCH – חישוב כתובת הפקודה הבאה לביצוע והבאת הפקודה הבאה מהזיכרון.
- DECODE - פיענוח הפקודה הנוכחית ומעבר הפקודה ליחידת הבקרה כדי שתדע איזה פקודה עלינו לבצע. בנוסף בשלב זה כבר נוציא רגיסטר מהזיכרון לצורך מימוש הפקודה.
- הפעולות ITYPE, JTYPE,RTYPE -





DATAPATH UNIT:

היחידה הנ"ל מממשת רכיב CPU בעלת BUS 2 באופן הבא:



המודל הנ"ל הינו הגוף של המערכת, כלומר הוא מקבל פקודה מהPROGRAM MEMORY ומבצע אותה בהתאם על ידי DECODER. לאחר פיענוח הפקודה נעביר את הסיגנל המתאים לקונטרול שידליק את בדגלים המתאים לביצועה ובעצם כך המערכת עובדת.

מטרת היחידה הזו הינה לבצע את הפעולות של המערכת בהן קיימות פעולות אסינכרוניות ואסינכרוניות.

<u>המערכת מורכבת ממספר רכיבים עיקריים:</u>

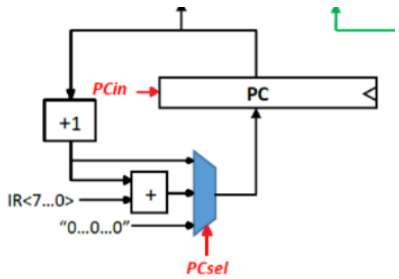
- PC - רכיב שמכיל את הכתובת של הפקודה הבאה שאמורה להתבצע. ה PC-משמש כמונה הוראות, והוא מתעדכן בכל מחזור שעון, כך שהמעבד תמיד יודע מאיפה להמשיך את ביצוע התוכנית.
- PROGRAM MEMORY – רכיב המקבל את הפקודות המבוקשות על ידי הכנסת INPUT מתאים ומעביר את הפקודה הבאה בתוכנית ל IR.
- IR – רכיב המקבל את הפקודה הנדרשת לפיענוח ומעביר את הOPC המתאים ואת הרגיסטרים הרצויים לקריאה או כתיבה, ובמידה ויש מעביר גם קבוע.
- OPC DECODER – הרכיב מקבל את הפקודה המבוקשת מהIR, מפענח אותו ומעביר ליחידת הבקרה את הסיגנל המתאים.
- REGISTER FILE- הרגיסטרים הקיימים במערכת, בסך הכל 16 רגיסטרים.
- DATA MEMORT – יחידת הזיכרון, יחידה השומרת את המידע בזיכרון כך שניתן לקרוא או לכתוב לזיכרון על ידי ברירה של MUX לביצוע הפעולה הנדרשת.
- ALU – היחידה האריתמטית בו מבוצעות הפעולות הנדרשות.

מעבר המידע במערכת מבוצע על ידי BUS 2 מה שמאפשר מעבר מידע מהיר יותר מאשר מערכת מבוססת BUS אחד, מה שמוסיף MUX למערכת לניתוב המידע המתאים ושליפתו מה BUS הרצוי.

-PC

רכיב שמכיל את הכתובת של הפקודה הבאה שאמורה להתבצע מתוך קובץ הטקסט הנכנס למעבד. ה PC-משמש כמונה הוראות, והוא מתעדכן בכל מחזור שעון, כך שהמעבד תמיד יודע מאיפה להמשיך את ביצוע התוכנית. את הPC נעדכן ב 3 דרכים:

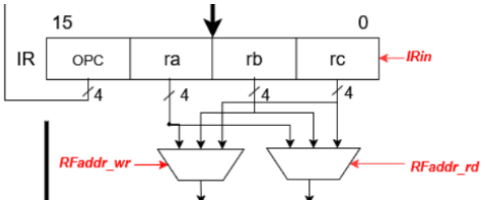
- PC+1 כאשר נרצה לעבור לפקודה הבאה ברשימה יתקבל כאשר PCSEL=01.
- PC+1+OFFSET כאשר יש צורך לעבור לפקודה הדורשת הוספת קבוע ל PC יתקבל כאשר PCSEL= 10.
- 0 וקטור כאשר יש צורך לאתחל את ה PC יתקבל כאשר PCSEL=11.



-IR

רכיב במעבד שמקבל את הפקודה לביצוע, מפענח אותה, ומוציא את קוד הפעולה (OPC) הנדרש, יחד עם כתובות הרגיסטרים שיש לקרוא מהם או לכתוב אליהם. במידת הצורך, הוא גם מעביר ערך קבוע כחלק מההוראה.

את הרגיסטרים נחלק על פי הביטים הרלוונטיים בצורה הבאה וכך נשתמש כל פעם ברגיסטר הרצוי:



-OPC DECODER

רכיב המקבל את 4 הביטים MSB של הפקודה וכך מעלה את הדגל המתאים המציין איזה פקודה עלינו לעשות במעבד.

הפקודה המתאימה עולה לCONTROL ובהתאם לFSM הפקודה תתבצע שלב אחרי שלב.

-ALU

היחידה האריתמטית של המערכת הפועלת לפי הOPC של הפקודה הנוכחית. המערכת מבצעת חיבור וחיסור באמצעות שרשור FA לכדי מחבר ופעולות XOR, OR, AND באמצעות שערים לוגיים.

בנוסף הרכיב מעלה דגלים CFLAG, NFLAG,ZFLAG בהתאם לפעולה שהתבצעה ב ALU בהתאם.

לבסוף נקבל את תוצאת הרכיב דרך היציאה C שיוצאת לbus A.

מימוש המערכת בקוד:

את המערכת מימשנו באופן הבא:

1. יצירת FSM למימוש הקוד.
2. יישום הרכיבים הפנימיים במערת ה DATAPATH כלומר, קובץ קוד המתאים ליישום PC, IR, PM,ALU, DM, RG , OPC DECODER .
   כתיבת מערכת המצבים ברכיב CONTROL, הדלקת סיגנלים רצויים המיצגים מעבר בין שלבים לביצוע הפקודה.
3. כתיבת שכבת הTOP של מערכת DATAPATH. כלומר, יצירת הWIRES המקשרים בין הרכיבים להעברת המידע במערכת, בניית MUX לבחירה בין חוטי המידע או יישום קריאה/כתיבה במערכת, בניית חוצצים השולטים בהגעת המידע לקו.
4. שכבת ה TOP שכבה במקשרת בין הDATAPATH לבין רכיב ה CONTROL למעבר פקודות מהמוח לגוף המערכת.

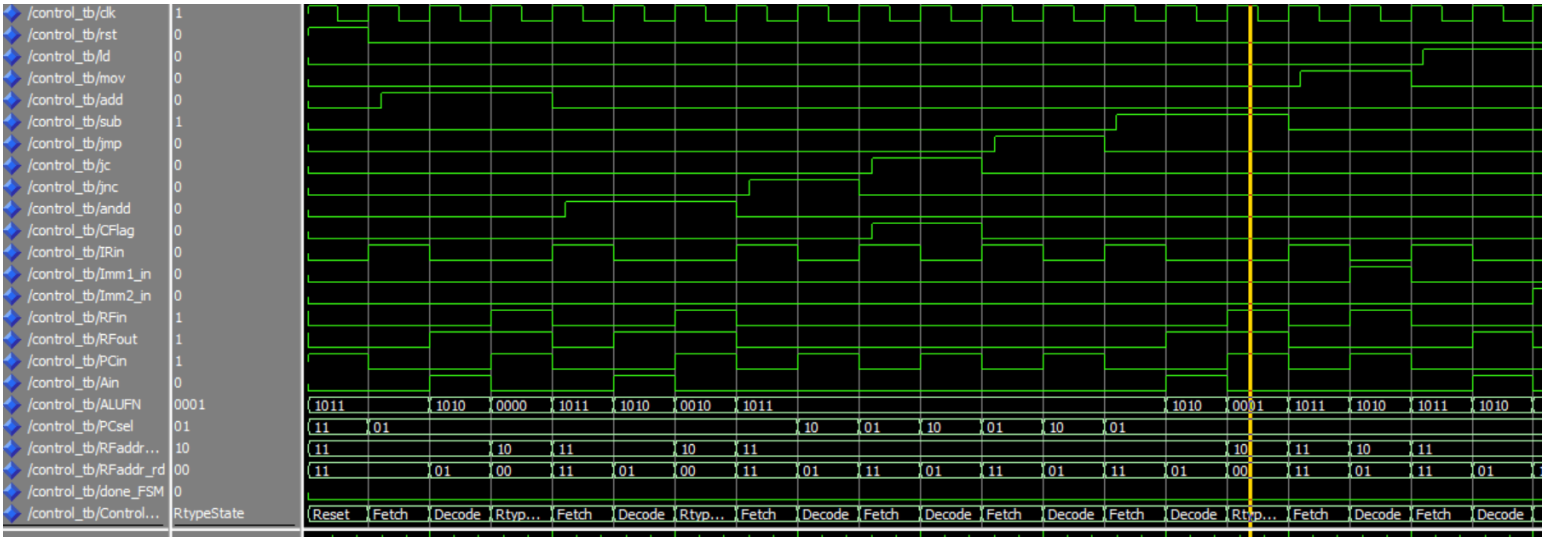
סימולציה:

סימולציה עבור הCONTROL UNIT

בסימולציה זאת נרצה לראות האם עולים הדגלים בהתאם לכל שלב במערכת המצבים ובהתאם לפקודה המבוצעת.

דגלים אלה הינם דגלים המתקבלים מהDATAPATH ליחידת ה CONTROL ועוברים את השלבים המתוארים בFSM שתכננו.

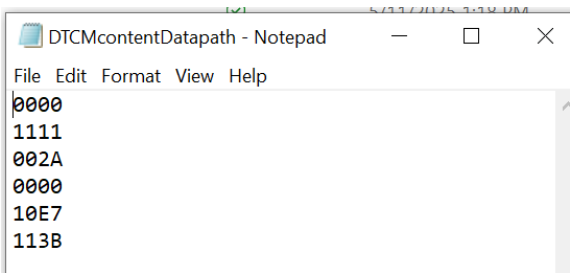
כדי לבצע פקודה נצטרך לדאוג שהגדלים הרלוונטיים עולים למשך כמות הcycles הנדרשים לביצוע הפקודה.

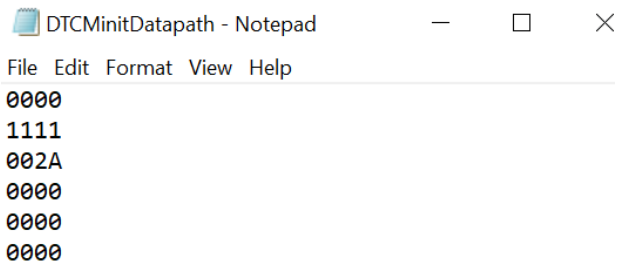


כפי שניתן לראות בדיאגרמת הגלים מופיעים בתחתית שלה מעברים בין מצבי המכונה שלנו בהתאם לפקודות הנתונות. לדוגמא עבור הסמן אנו נמצאים על הסיקל השלישי בפעולת חיסור בו מצב המכונה הוא Rtypestate כלומר השלב בו הALU מבצע את פעולת החיסור עצמה ומאחסן ברגיסטר המתאים לפי הפקודה. לכן בנקודת זמן זו נראה את הסיגנלים : RFout,PCin,RFin עולים ל-'1' ובהתאם גם הסיגנלים שאחראים על כתובות הקריאה והכתיבה לRegfile משתנים בהתאם לסמן על רגיסטרים: rc,ra.

סימולציה עבור ה DATAPATH

בסימולציה זאת נרצה לראות האם המעבר בין הרכיבים של המערכת הנ"ל מיושם נכון במערכת שלנו. נבדוק זאת על ידי כך שעלו הדגלים הנכונים לפי ה FSM של קווי הבקרה.





כפי שניתן לראות התוכנית שלנו מבצעת שלוש פעולות בין הערכים השמורים בתאים 1,2 כאשר לפני האחסון היא בודקת האם קיים carry לפעולת החיבור האחרונה ולכן אינה מאחסנת דבר בתא

הרביעי בזיכרון ובתאים 5,6 מאחסנת את פעולות החיסור ו-xor בהתאמה. מצורפות תמונות הקבצים שמופיעים גם כאן בתיקייה files המצורפת.

סימולציה עבור המערכת כולה

סימולציה בה מכניסים קובצי טקסט בינארי למערכת כפי שהסברנו במבוא. לאחר האתחול נריץ סט פקודות בהתאם לקובץ הפקודות שהכנסנו המותאמות ל ISAשלנו ונבדוק האם הן מתבצעות כנדרש.

אופן ביצוע הבדיקה הינו השוואה בין הOUTPUT הניתן לנו לבין הOUTPUT שקיבלנו לאחר הרצת התוכנית במעבד שלנו.

מצורף בעמוד הבא תוכן הזיכרון לפני הפעולות מימין ומשמאל תוצאתיו של זוג רשימות הפוכות מ-0 עד 14 ופעולת ה-xor בין כל זוג ערכים תואמים ברשימה(כלומר המשלים ל14).

ניתן לראות את תוצאות התוכנית באופן ברור בקבצי ה files המצורפים.

0000	0000
0001	0001
0002	0002
0003	0003
0004	0004
0005	0005
0006	0006
0007	0007
0008	0008
0009	0009
000A	000A
000B	000B
000C	000C
000D	000D
000D	000D
000C	000C
000B	000B
000A	000A
0009	0009
0008	0008
0007	0007
0006	0006
0005	0005
0004	0004
0003	0003
0002	0002
0001	0001
0000	0000
000D	0000
000D	0000
0009	0000
0009	0000
000D	0000
000D	0000
0001	0000
0001	0000
000D	0000
000D	0000
0009	0000
	0000