דו"ח מעבדה- A Simple Processor

מגישים : לירן גולן 311121073

דניאל מרג'י 342533627

הסבר כללי של המעבד

המעבד הוא רכיב חומרתי המבצע פעולות המתקבלות מכניסה של 9 ביטים מ-Din

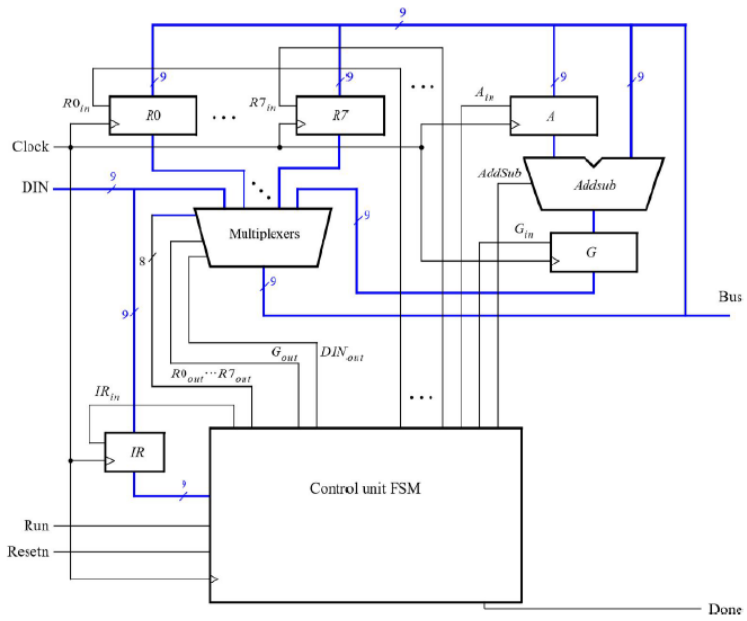
המעבד בנוי ממספר רכיבים הכוללים(פירוט בקצרה של הרכיבים):

- אוגרים – במעבד שלנו ישנם 11 רכיבי זיכרון $R_0, R_1 \dots R_7, IR, A, G$ כאשר כל יח' זיכרון שומרת בתוכנה מידע בעל 9 ביטים.
- ALU – יחידת החישוב המבצעת פעולות אריתמטיים על פי אות הבקרה המתקבל. דוגמה חיבור/חיסור בין 2 ביטים.
- Mux – המרבב, מטרתו היא לברור את הערוץ המתאים על פי האות המתקבל, כך נוכל לעביר מידע מDIN או מרגיסטר מסוים אל הBUS (חוט העברת המידע מכיל 9 ביטים) ודרכו נוכל להעביר את המידע אל רגיסטר אחר.
- FSM – הוא רכיב המוציא את האותות הבקרה הרלוונטיים בהתאם לכל פקודה ובהתאם למחזור של הפקודה.

במעבדה זו מימשנו מעבד מסוג multi cycle processor כאשר כל פקודה מחולקת למספר cycles שיש לבצע עד להשלמתה. כאשר בכל שלב בפקודה מתבצעת מחזור שעון יחיד.

דיאגרמה של המעבד

אנו מימשנו את המעבד הניתן לנו בתרגיל מהתצורה הבאה:



כאשר אנו מימשנו את המערכת באמצעות שלושה קבצי Verilog

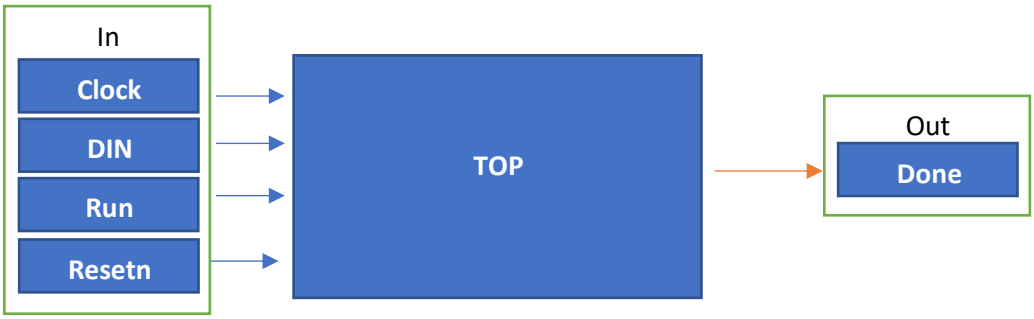
כאשר הTOP מוכלל בתוך הקובץ proc.v שהוא מבוסס על skeleton הנתון.

הכולל את יחידות: ALU,MUX,FSM

אשר ייבא אליו את האוגרים regn.v

ואת רכיב dec3to8.v

הסבר כניסות ויציאות של הTOP



הבלוק של TOP מתאר את המבנה הפנימי של המעבד.

נסביר כעת, על הכניסות והיציאות של רכיב זה.

כניסות:

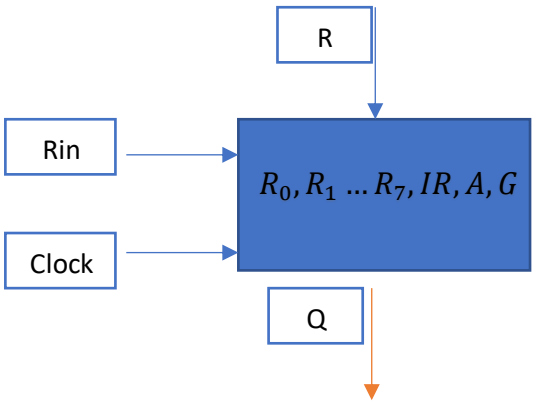
- Clock – השעון של המעבד.
- Din - המידע שנכנס אל המעבד, שאותו יתרגם ויצטרך לבצע.
- Run – כאשר הוא נמצא ב-1 , המעבד יכול לקבל פקודה חדשה לביצוע.
- Resetn – מאתחל את המעבד.

יציאות:

- Done – מסמל סיום ביצוע הפקודה (כאשר Done=1)

הסבר כניסות ויציאות של הרכיבים השונים:

האוגרים - $R_0, R_1 \dots R_7, IR, A, G$ – כמו שאמרנו, תפקיד הרגיסטר הוא לשמור את המידע המאוחסן בו. כאשר במקרה של A/G מטרה היא לשמור את המידע באופן זמני לצורך חישוב או העברת המידע.



נסביר כעת, על הכניסות והיציאות של רכיבים באופן כללי .

כניסות:

- Clock – השעון של המעבד.
- Rin – מאין enable, אות בקרה המאפשר כתיבה אל הרגיסטר.
- R – תוכן שאותו אנו רוצים לשמור ברגיסטר.
  הערה: שמירה של מידע חדש יתבצע רק כאשר Rin=1 וגם יש עלית שעון.

יציאות:

- Q – תוכן הרגיסטר המעודכן.

הערה: עבור סוג רגיסטר שונה, נתנו שמות שונים בהתאמה, לדוגמה עובר IR בכניסה הוא מקבלDIN , וביציאה IR , עבור A מקבל BUS וביציאה מוציא Aout ועבור G מקבל כניסה של AddSubOut ומוציא Gout, עבור $R_0, R_1 \dots R_7$ מקבל ככניסה BUS ומוציא $R_0, R_1 \dots R_7$

יחידת הבקרה – FSM תפקיד היחידה היא לנהל את פעולות המעבד ע"י קריאה הפקודה ושליחת אותות הבקרה המתאימים על פי מחזור השעון.

כניסות:

- Clock – השעון של המעבד.
- IR – 9 ביטים של הפקודה המתקבלת מהמוצא של הרגיסטר IR
- Run – כאשר הוא נמצא ב-1 , המעבד יכול לקבל פקודה חדשה לביצוע.
- Resetn – מאתחל את המעבד.

יציאות:

- Done – מסמל סיום ביצוע הפקודה (כאשר Done=1).
- IRIN – מאין enable, אות בקרה המאפשר כתיבה ושמירה של הפקודה.
- Rout- wire בעל 8 ביטים המשמש כסלקטור עבור ה-multiplexers כאשר רוצים לגשת לאחד מן הרגיסטרים.
- Gout-סלקטור נוסף עבור הרגיסטר G.
- DINOUT-סלקטור נוסף כאשר הפקודה דורשת פעולה עם קבוע.
- Gin/Ain/Rin-אות בקרה , כאשר הערך של האות בקרה שווה ל 1 ניתן לכתוב לרגיסטר.
- ADDSUB-מגדיר את סוג הפקודה שצריך לבצע.

AddSub - היחידה הזו מוכללת בתוך רכיב הproc . הסבר על היחידה על פי השרטוט המקורי, היחידה מקבלת 2 רגיסטרים ו3 קווי בקרה מFSM אשר קובעים את סוג הפעולה האריתמטית שיש לבצע, הפעולות האריתמטיות הבסיסיות הם חיבור וחיסור. היחידה מבצעת ושומרת את התוצאות.

כניסות:

- A – מייצג את רגיסטר RX
- B – מיצג את הרגיסטר RY אות הקובע את din
- ADDSUB – מציין את סוג הפקודה שיש לבצע בין הרגיסטרים.

יציאות:

- Alu_out – התוצאה המתקבלת מהחישוב בהתאם לסוג הפעולה האריתמטית.

Dec3to8 – היחידה אחראית להעביר את הייצוג של רגיסטר מסוים מ3 ביטים ל8 ביטים.

כניסות:

- W – אות כניסה באורך של 3 ביטים.
- En – כאשר En=1 הרכיב פועל.

יציאות:

- Y – אות המוצא שמכיל את הקידוד המתאים באורך של 8 ביטים.

Multiplexer – קווי הבקרה של הבורר המתקבל מFSM ועל פיהם, הוא מעביר את התוכן הרגיסטר הנבחר אל Buswires שהוא באורך של 9 ביטים.

כניסות:

- R[0...7] – תוכן הרגיסטר הi לדוגמה R1, זה התוכן של אוגר 1R
- Din – המידע שנכנס אל המעבד
- G – תוכן רגיסטר G
- Rout- wire בעל 8 ביטים המשמש כסלקטור עבור ה-multiplexers כאשר רוצים לגשת לאחד מן הרגיסטרים
- Gout – ביט יחיד, המשמש כסלקטור עבור הmultiplexerx כאשר רוצים לגשת לG
- Dinout - ביט יחיד, המשמש כסלקטור עבור הmultiplexerx כאשר רוצים לגשת לD

יציאות:

- Buswire – אות המוצא שמכיל את הקידוד המתאים באורך של 9 ביטים.

פקודות של המעבד:

המעבד מקבל 9 הוראה בגודל של 9 סיביות:

Y	Y	Y	X	X	X	I	I	I

פירוט:

- III-מייצג את סוג הפעולה שצריך לבצע לפי הפקודות.
- XXX- מתייחס לרגיסטר RX
- YYY- מתייחס לרגיסטר RY

כל פקודה ניתנת לחלוקה על פי סוגה כאשר כל שלב מבצע מחזור שעון אחד, מספר מחזור שעון המינימאלי הוא 2 והמקסימום ה4 פקודות שונות.

נרצה לתאר את הפקודות על פי מחזורי השעון, נקרא למחזור הראשון $T0$, המחזור השני $T1$ וכן עבור המחזור הרביעי $T3$.

המחזור הראשון מופיע בכל הפקודות (ולכן נרשום אותה פעם אחת), אותות הבקרה משתנים כתלות בפקודה ולאחר מכן מתאפסים.

<u>פירוט מחזורי שעון על פי:</u>

T0- המעבד קורא את הפקודה שהתקבלה וההוראה נשמרת ברגיסטר **IR**

<u>פקודת MV:</u> הפקודה מעתיקה את המידע הנמצא ברגיסטר RY לתוך RX

T1- היציאות שמגיעות לmux מעבירות את תוכן הרגיסטר RY דרך Buswire , אות הבקרה של RXin=1 , כעת ניתן לכתוב לRX דרך מידע שנמצא בBusWire. כאשר הפעולה מסתיימת אות הבקרה יעלה ל-1 כלומר, Done=1

<u>פקודת MVI</u> הפקודה מעתיקה את המספר הקבוע לתוך RX

T1- היציאות שמגיעות לmux מעבירות את תוכן הרגיסטר DIN דרך Buswire , אות הבקרה של RXin=1 , כעת ניתן לכתוב לRX ודרך המידע שנמצא בBusWire. כאשר הפעולה מסתיימת אות הבקרה יעלה ל-1 כלומר, Done=1

<u>פקודת SUB/ADD</u> מבצע חיבור/חיסור של תוכן מרגיסטר RX עם RY ושמירת התוצאה ברגיסטר RX

T1- היציאות שמגיעות לmux מעבירות את התוכן RX דרך הBusWire, כאשר אות הבקרה $Ain = 1$ המאפשר שמירת נתונים של ברגיסטר A.

T2- היציאות שמגיעות לmux מעבירות את התוכן RY דרך הBusWire, לאחר מכן, יתבצע פעולה אריתמטית (חיבור או חיסור) באמצעות ALU בין תוכן רגיסטר A לבין RY שנמצא בBus אות הבקרה $Gin = 1$ מאפשרת שמירת נתונים ברגיסטר G.

T3- המידע מרגיסטר G עובר דרך הBuswire דרך הmux כאשר אות הבקרה $Rxin = 1$ מאפשר לשמור את המידע הנמצא בBUS אל RX , כאשר הפעולה מסתיימת אות הבקרה יעלה ל-1 כלומר, Done=1

<u>פקודות ADDI/SUBI:</u> מבצע פעולת חיבור של מספר קבוע לרגיסטר RX ושמירת התוצאה בRX

T1- היציאות שמגיעות לmux מעבירות את התוכן RX דרך הBusWire, כאשר אות הבקרה $Ain = 1$ המאפשר שמירת נתונים ברגיסטר A.

T2- היציאות שמגיעות לmux מעבירות את התוכן DIN דרך הBusWire, לאחר מכן, יתבצע פעולה אריתמטית (חיבור או חיסור) באמצעות ALU בין תוכן רגיסטר A לבין המידע שנמצא בBus אות הבקרה ידלוק כלומר $Gin = 1$ מאפשרת שמירת נתונים ברגיסטר G.

T3- המידע מרגיסטר G עובר דרך הBuswire דרך הmux כאשר אות הבקרה $Rxin = 1$ מאפשר לשמור את המידע הנמצא בBUS אל RX , כאשר הפעולה מסתיימת אות הבקרה יעלה ל-1 כלומר, Done=1

כעת נשלים את הטבלה מהקובץ:

פקודה	קידוד	T0	T1	T2	T3
MV	000	IRIN	RYout,RXin,Done		
MVI	001	IRIN	DINout,RXin,Done		
ADD	010	IRIN	RXout,Ain	RYout,Gin	Gout,RXin,Done.
SUB	011	IRIN	RXout,Ain	RYout,Gin	Gout,RXin,Done.
ADDI	100	IRIN	RXout,Ain	DINout,Gin	Gout,RXin,Done.
SUBI	101	IRIN	RXout,Ain	DINout,Gin	Gout,RXin,Done.

סימולציות:

נרצה לבצע סימולציות הבודקות את תקינות המעבד, לשם כך נבחר מקבץ של סימולציות הבוחנות את הפקודות שהגדרנו עבור רגיסטרים מסוימים. נציין שלא נוכל לכסות את כל המקרים, ובכל טסט לא נוכל לוודא שהבדיקה מתקיימת עבור כל רגיסטר.

נרצה להניח מספר הנחות:

- הפעולה תתבצע עבור כל קומבינציות הרגיסטרים למרות שנבחן רק מקרים בודדים בהסתברות אקראית.
- נבחן רק מספר ערכים קבועים מסוימים (Din) ונניח שהפעולה תקינה עבור כל המספרים האפשריים.

בנוסף, ישנם מקרים שלא נוכל לכסות ולכן נרצה להתרכז במספר סימולציות אשר ייבחנו מקרים רבים, המתארים בצורה טובה את תקינות המעבד.

הבדיקות שבחרנו לבצע:

- בדיקת Reset
- בדיקת Overflow
- בדיקה של כל הפעולות הבסיסיות
- בדיקה של הפעולות שבחרנו להוסיף למעבד – Addi, Subi
- בדיקה של העברית מידע בין כל הרגיסטרים ע"י שימוש בפקודת movi עבור כל רגיסטר.
- בדיקת RUN
- בדיקת timing
- בדיקת הרבה פעולות ברצף.

בדיקות שלא בדקנו

- פקודות אסורות נוספות
- בדיקה של מקרי קצה, כמו למשל פתיחה של 2 רגיסטרים ביחד $(Rin_1\ ,\ Rin_0)$
- מעבר על כל המעברים אפשריים עבור בדיקה ספציפית.

בדיקה של כל הפעולות הבסיסיות

- תחילה נבדוק את פקודות ה-movi ו-add

נתחיל עם סימולציה פשוטה : נכניס ב רגיסטר 1 ערך 4 וברגיסטר 2 את ה ערך 8 ע"י קוד פקודה 001 (**movi**) .

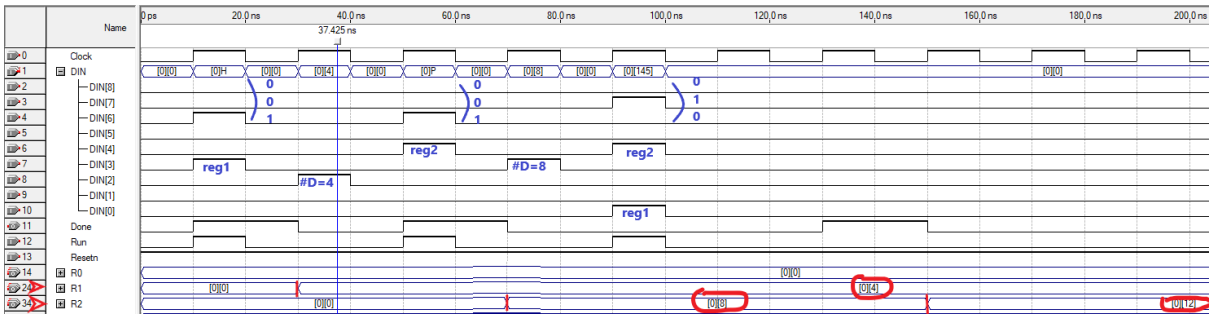
הערה טכנית אך חשובה :באיורים של הטסטים משמסומן בכחול הוא input ובאדום output.

אפשר לראות בסימולציה (באדום) שהערכים מתעכנים מחזור שעון אחרי.

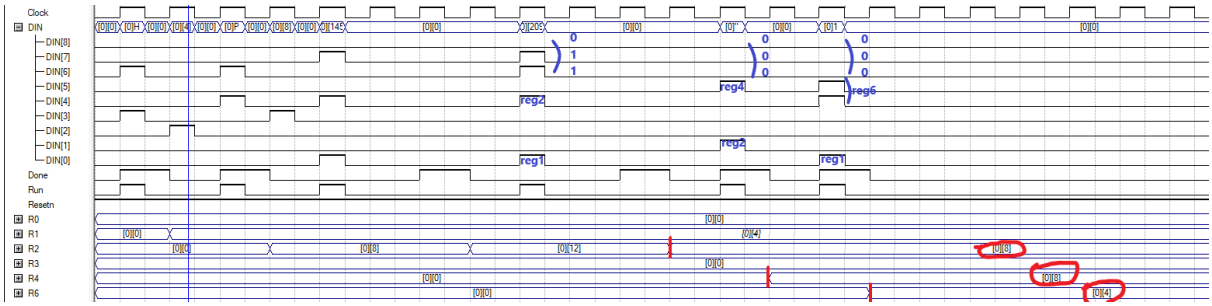
כמובן בערך #D מוכנס ב cycle הבא אחרי הפקודה , כאשר לחצנו על Run .

עתה נסכום R1 ו R2 ו נכניס את התוצאה ב R2 . נעשה את זה על ידי קוד פקודה 010 (**add**) , כאשר R2 מוכנס שדה ה XXX ו R1 מוכנס בשדה ה YYY .

אשפר לראות ש 12 נכנס בהצלחה ב R2 אחרי 4 cycles .



נמשיך להפעיל את ה מעבד שלנו:



- נבדוק את פקודות ה-sub ו-mov

עתה נרצה לבצע חיסור בין R2 ל R1 ו להכניס את התוצאה ב R2 . נעשה את זה ע"י קוד פקודה 011(sub).

ניתן לראות כי אחרי 3 מחזורי שעון הערך 8= 4-12 נכנס בהצלחה ל R2 .

עתה נרצה להעביר את תוצאות החישובים שלנו ל רגיסטרים אחרים : ניקח R4 ו R6 .

נעשה את זה ע"י פקודה 000 (mov) כאשר מכניסים את מה שב R2 (הערך 8) ב R4 ומחכים מחזור שעון כדי להכניס את את מה שב1R (הערך4) ב R6 . ניתן לראות כי העברות האלו נעשו בהצלחה : הערכים

ברגיסטרים שמסומנים עיגולים אדומים מתעכנים במחזור אחרי הפקודה.

- <u>בדיקה של הפעולות שבחרנו להוסיף למעבד – Addi, Subi</u>

עתה נרצה לבדוק את הפעולות addi ו subi .

מה שחשוב להבין זה ש בניגוד לפעולה movi ה #D יוכנס ע"י המשתמש רק ב T2.

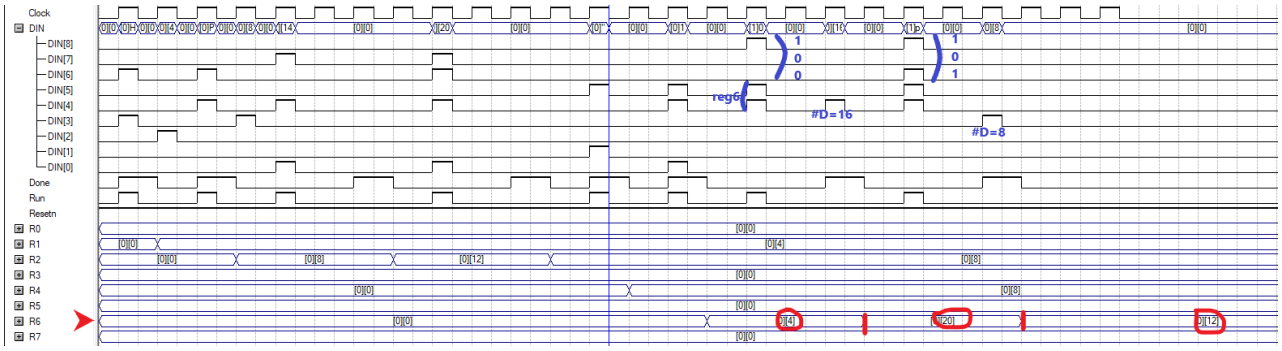
Gout,RXin,Done.	DINout,Gin	RXout,Ain	IRIN	100	**ADDI**
Gout,RXin,Done.	DINout,Gin	RXout,Ain	IRIN	101	**SUBI**

כפי שניתן לראות בטסט למטה, המתייחס לבדיקת ADDI SUBI

נתבונן בחץ האדום, בריסטר R6 , נשים לב שיש בו כבר את הערך 4, נרצה להעלות את הערך ב-16 עם פקודת addi (100) כאשר 16 מוכנס לא במחזור הבא אחרי הפקודה אלה המחזור אחרי ורואים שמתקבל 4+16=20 ב R6.

בנוסף מפעילים עליו פקודה subi (101) כאשר אנחנו רוצים להוריד את הערך ב- 8 ומקבלים בסוף התהליך את הערך החדש הנמצא ב R6 שהוא 20-8=12.

כמובן ביצענו את כל הפעולות שתארנו לפני ברצף לפני פעולות אלו כדי לבדוק אם אין תופעות מוזרות ש קוראות.(**בדיקה של פעולות ברצף**)

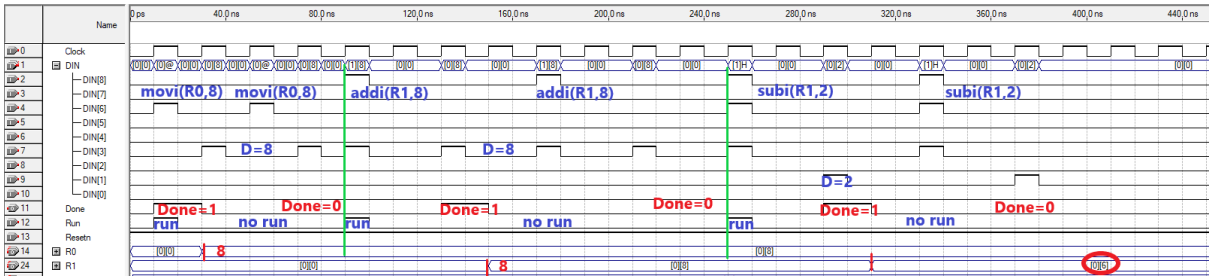


- <u>בדיקת RUN</u>

נוכל גם לבדוק שלא מתבצעת פקודה אם RUN לא דלוק :

ביצענו שלושה פקודות שונות : movi , addi , subi כאשר כל אחת מתבצעת פעמיים : פעם עם RUN ופעם בלי. באיור שמופיעה למטה מתאים לשלושה איזורים מופרדים ע"י הקווים הירוקים כאשר עבור כל אזור מתבצעת פעולה עם RUN תחילה ואז משכפלים את אותה פקודה אך בלי ה RUN.

ניתן לראות באיור כי אם המשתמש אינו מעלה את RUN אז Done יהיה 0 והערך לא מתעדכן:



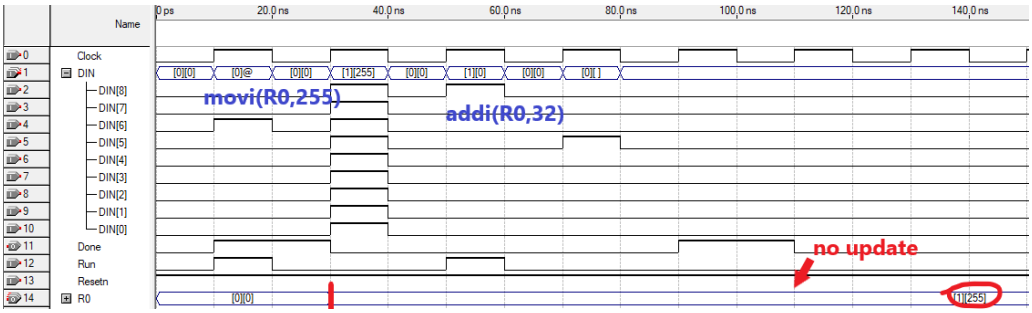
- <u>בדיקת Overflow</u>

עתה נשאל מה לגבי overflow ?

לכל רגיסטר יש 8 ביטים ולכן הערך המקסימאלי יהיה $2^8 - 1 = 255$.

אם ננסה להוסיף מה יתקבל ?

בצענו movi(R0,255) ואז ניסינו לעבור את ערך זה דרך פקודת addi(R0,32) :



התקבל שהרגיסטר אינו מתעדכן : אין למערכת שלנו אפשרות לתפוס error , הפעולה פשוט לא מתבצעת שזה כבר טוב.

עכשיו נוכל גם לשאול מה יקרה אם ננסה לבצע חיסור שאמור לתת ערך שלילי?

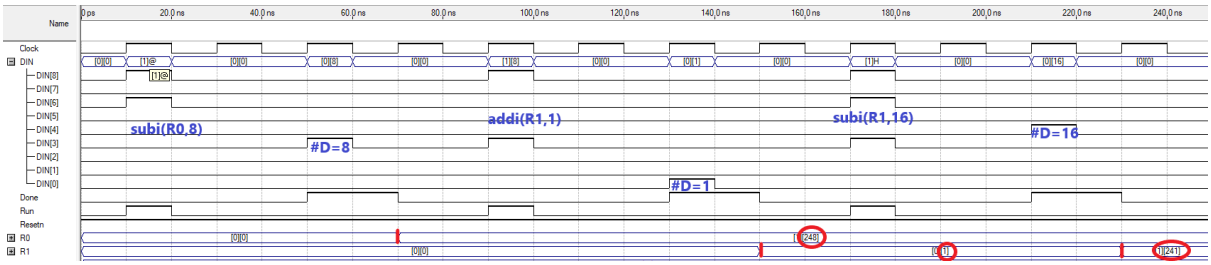
בשביל לבדוק את זה נבצע נבחן שני מקרים :

- המקרה ההתחלתי בו כל הרגיסטרים אמור להיות מאופסים
- מקרה כללי תוך כדי ריצת רצף פקודות

נתחיל במקרה ההתחלתי וננסה להחסיר 8 מR0 שמאותחל ב0 , נבצע subi(R0,8)

מתקבל כי הערך 248 נכתב ברגיסטר. זה ה overflow : לרדת מתחת לאפס זה לרדת ל 255 ומשם ממשיכים להחסיר כרגיל ולכן התקבל 255-7=248

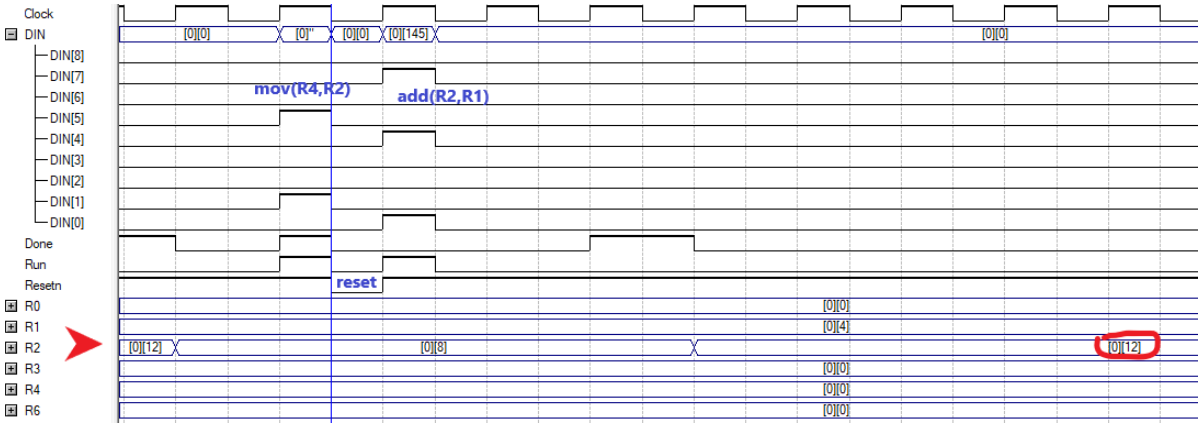
בשביל לבדוק את המקרה השני נבצע movi(R1,1) כדי לא להיות במצב התחלתי ונחסיר ערך גדול מ1 שזה יהיה 16 ע"י פעולת subi(R1,16). במקביל לתוצאה הקודמת מתקבל 255-15=241 .



ראינו שבכל מקרה של overflow המערכת שלנו אינה קורסת שזה מתאים לדרוש.

- <u>בדיקת Reset</u>

עתה ננסה ללחוץ על reset תוך כדי פעולה :

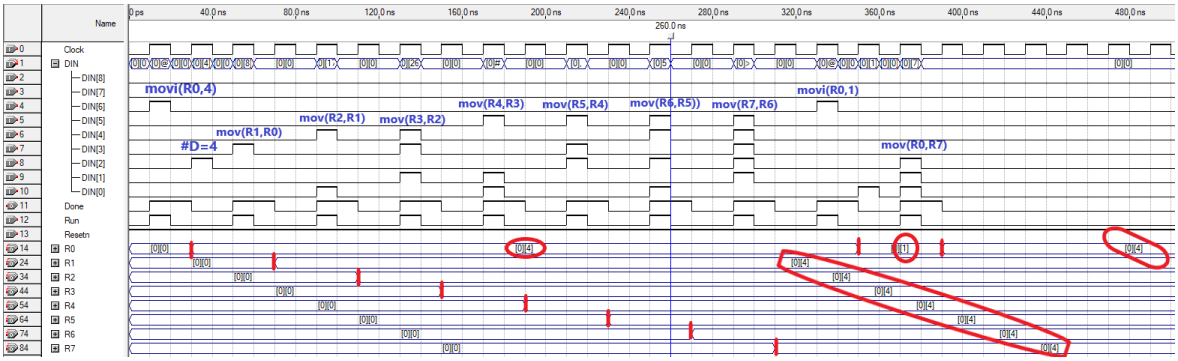


כפי שניצן לראות התחלנו את להכניס את R2 ב R4 אבל לחצנו על reset .

כתוצאה מכך יכלנו ישר (כלומר ב מחזור הבא) להכניס פעולה אחרת שהיא להוסיף את הערך של R1 ל R2 . ניתן לראות כי זה נעשה(R2 = 8 +4) בהצלחה בזמן הדרוש בלי שתיהיה השפעה מהפקודה הקודמת.

- <u>בדיקה של העברית מידע בין כל הרגיסטרים לייצוג תפקוד תקין של כל הרגיסטרים</u>

עכשיו נבדוק שכל הרגיסטרים ניתנים לכתיבה \ קריאה ע"י הכנסה של ערך כלשהו ב R0 (בחרנו 4) דרך הפקודה movi(R0,4). עכשיו נעביר ערך זה עם פקודת mov (000) מרגיסטר זה לR1 ומ R1 ל R2... עד שהערך מגיע לR7. היות ובערך 4 כבר נמצא ב R0 בחרנו לבצע movi נוסף לו עם ערך 1 movi(R0,1) : זה שהיינו מעבירים ל R0 את הערך 4 לא היה מוכיח שהצלחנו לכתוב בו כי ערך זה כבר היה שם. אחרי שהערך R0 התעדכן ל1 ביצענו את ה mov האחרון מ R7 ל R0 ורואים ש R0 כן מתעכן ל 4 :



כרגע למעשה הראנו ש:

- פעולות שונות של כמה רגיסרים תקינות ולא מפריעות אחת לשנייה
- הכתיבה \ קריאה של כל רגיסטר תקינה

מכאן היות ואי אפשר לבצע את כל הפעולות האפשריות בין כל הרגיסטרים כי יש הרבה הרבה אפשרויות וזה ייקח הרבה זמן נוכל להניח כי הם מתבצאות בצורה תקינה עבור קלט תקין .

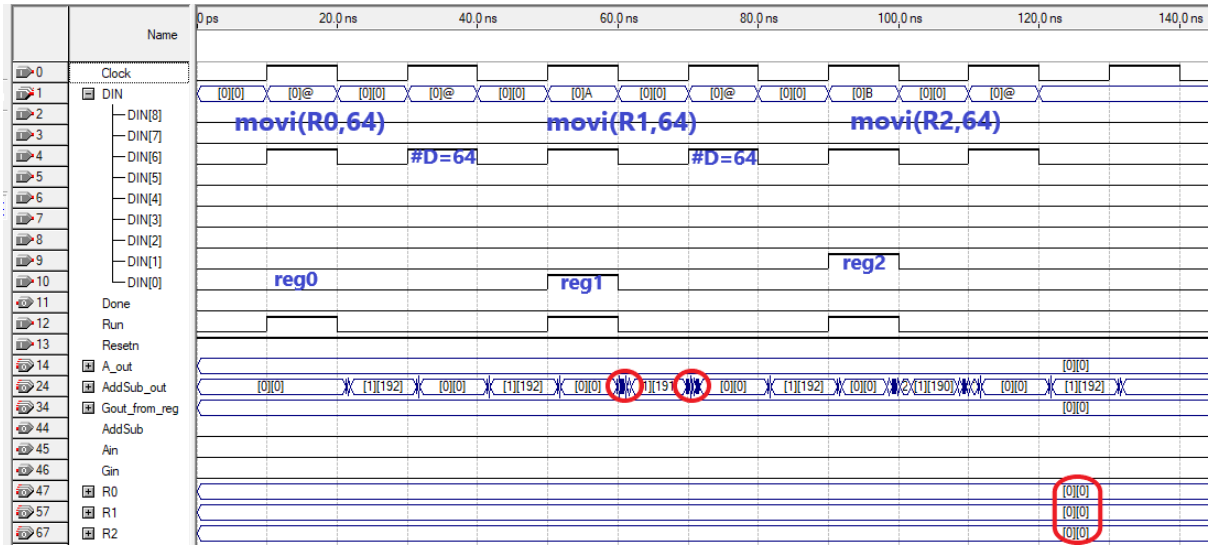
- <u>בדיקת timing</u>

ניתן גם לראות כי התדר המקסימלי של השעון של המעבד שלנו הוא $78.34\ MHz$:

	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock
1	Worst-case tsu	N/A	None	4.687 ns	DIN[6]	regn:reg_G|Q[7]	--	Clock
2	Worst-case tco	N/A	None	20.272 ns	Tstep_Q[0]~reg0	AddSub_out[6]	Clock	--
3	Worst-case tpd	N/A	None	11.929 ns	DIN[3]	AddSub_out[6]	--	--
4	Worst-case th	N/A	None	0.728 ns	DIN[1]	regn:reg_IR|Q[1]	--	Clock
5	Clock Setup: 'Clock'	N/A	None	78.34 MHz (period = 12.765 ns)	Tstep_Q[0]~reg0	regn:reg_G|Q[7]	Clock	Clock

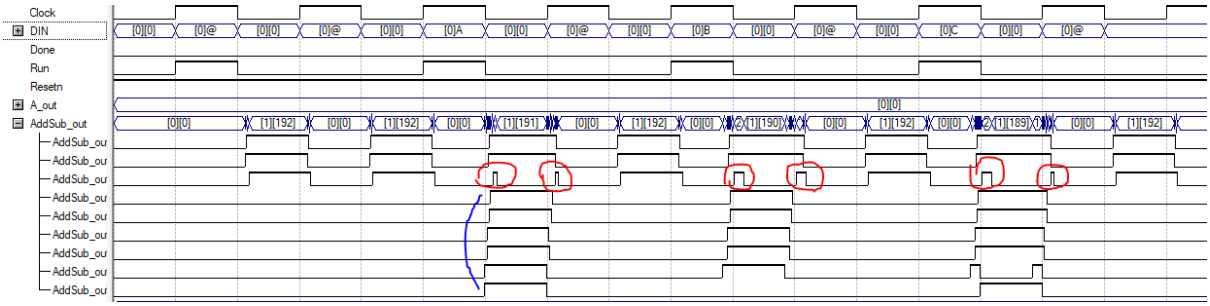
התיעצנו אם סטודנטים אחרים והיגענו למסקנה שזה יחסית טוב. אבל מצד שני אין לנו את אותם פקודות אז אי אפשר לדעת באופן מדויק. אנחנו מאמינים שאפשר לשפר אותו עבור ממוש שונה בקוד , ששונה מהAPI שנתנו לנו כי בקוד שלנו יצא שכמה שורות חזרו על עצמם.

כלומר אם מבצעים סימולציית Timing עם שעון של 20 ns לדוגמה :



ניתן לראות כי הרגיסטרים R0,R1,R2 אינם מתעדכני ונשארים באפס למרות שמנסים להכניס בהם 64.

העיגולים האדומים מראים לנו קפיצות חשודות נסתכל בכל הביטים של רגיסטר זה :



דבר ראשון שאפשר לראות (בכחול) זה שהביטים לא מתעכנים כולם בבת אחת : הראשון הוא התחתון וככל שיש לנו יותר ביטים , לביט האחרון יהיה delay גבוה.

ניתן לראות כי נוצרת בעיית timing אורך כל זמן פעולת המעבד : הביט השישי (באדום) עובר קפיצות לא רצויות.

כמובן כל הבדיקות האלו אינן מכסות את כל המקרים ואינן מבטיחות לנו שהמעבד שלנו יתפקד כראוי.

אם היינו רוצים ליהות יותר בטוחים היינו צריכים להריץ מלא סימולציות אבל באיזה צורה היינו עושים זאת ?

לבדוק את כל הקלטים באופו נפרד זה אפשרי כי יש רק 8 ביטים של קלט עבור כל פקודה אז היינו בודקים אם פקודה לא קיית (תקינה) לא משפיעה על פקודה תקינה . אפשר בפרט לבדוק שהרגיסטרים אינם משתנים עבור פקודה כזאת.עכשוו לגבי שרשור של פקודות כמובן יש אינסוף אפשרויות ולכן היינו מבצעים סימולציות באופן אקראי בתחומים "המסוכנים" : היות והמטרה שלנו היא לתפוס שגיעה הדבר הנכון ביותר זה קודם כל לבדוק שאין שגיאה איפה שיש את ההסתברות הכי גבוהה תיהיה שגיעה ולכן ננסה למפות את כל המערכת שלנו ולזהות תחומים מסוכנים. עבור בדיקת פונקצינליות זה יוותר קשה לזהות תחומים אלו אבל בדיקת תזמון אפשר להסתכל אפה שאין למערכת margin בבחינת זמן.

דיון – איזה מעבד היינו בוחרים.

<u>השוואה בין multi ל-single ומה היינו משנים</u>

קודם כל כפי שאמרנו בהתחלה המעבד שלנו הוא multi cycle ,אם נשווה אותו ביחס לsingle cycle , אופן העבודה בmulti cycle הוא שונה, מאחר שכל פקודה לוקחת מספר מחזורי שעון , כאשר בכל שלב בפקודה מתבצע במחזור שעון אחד, לעומת הsingle כאשר כל פקודה מבוצעת במחזור שעון יחיד בעל זמן מחזור ארוך מאוד.

<u>יתרון ב-single</u>

הוא פשוט לתכנון

<u>חסרון</u>

חוסר יעילות מאחר שזמן כל הפקודות הוא מחזור שעון אחד, שנקבע עפ"י הפקודה הארוכה ביותר, ללא תלות במה שהן באמת עושות.

מעבור multi – הוא בעצם "שיפור" של המעבד single **כל הפקודה לוקחת מספרי מחזורי שעון** – כאשר בכל שלב בפוקדה מבצע מחזור שעון אחד. [בגלל שעבדנו עם multi אז ה-design כולל רגיסטרי ביניים שיקפיאו את המצב אחרי כל שלב כמו רגנסטר לשמירת ALU ,רגיסטר A ...] ולכן אם היינו רוצים לעבוד עם single לא היינו צריכים רגיסטרים הללו. בנוסף היינו בוחרים את הפקודה עם הזמן הארוך ביותר, והיינו יכולים לשנות את design ולכן ה skeleton שנתנו לנו ישתנה.

<u>יתרון mutli</u>

- לפקודות שונות ייקח זמן שונה לרוץ – פקודה שמשתמשת בפחות שלבים, תיקח פחות זמן.
- המימוש מאפשר שכל יחידה תתפקד יותר מפעם אחת עבור פקודה על עוד זה מתבצע במחזורי שעון אחרים.

<u>חסרון</u>

תכנון מורכב יותר.

השוואה בין multiל- pipline

נשים לב, שאם היינו עובדים במעבד מסוג Pipeline-שלבים שונים של פקודות שונות מתבצעים במקביל, ובכך בעקבות המקביליות, בממוצע היינו מקבילים תוצאות טובות יותר מאשר מעבד multi ולכן היינו בוחרים לעבוד בו.

<u>ההבדלים:</u>

בעיקרון יש מספר רב יותר של הבדלים, אבל נתייחס לחלקם.

- בארכיטקטורת ה-pipeline מספר מחזורי השעון עבור כל פקודה

יהיה קבוע -5 ,בניגוד לארכיטקטורת ה-cycle multi בה מספר מחזורי השעון עבור כל פקודה הוא שונה, 3-5.

- בארכיטקטורת ה-pipeline בכל מחזור שעון יוצאת פקודה חדשה, בניגוד לארכיטקטורת ה-cycle multi ,בה כל פקודה רצה בנפרד על המעבד ורק כשסיימנו לבצע פקודה יוצאת הפקודה הבאה.
- בארכיטקטורת ה-pipeline לא ניתן להשתמש במשאבי מחשוב בו זמנית ובשלבים שונים, בניגוד לארכיטקטורת ה-cycle multi.

הערה: אם היינו רוצים לממש pipeline בעתיד שהיינו צריכים לזהר מ- Hazards. בשביל לעשות את כל זה היינו צריכים לשנות את ה design ואת ה- api שנתון לנו.