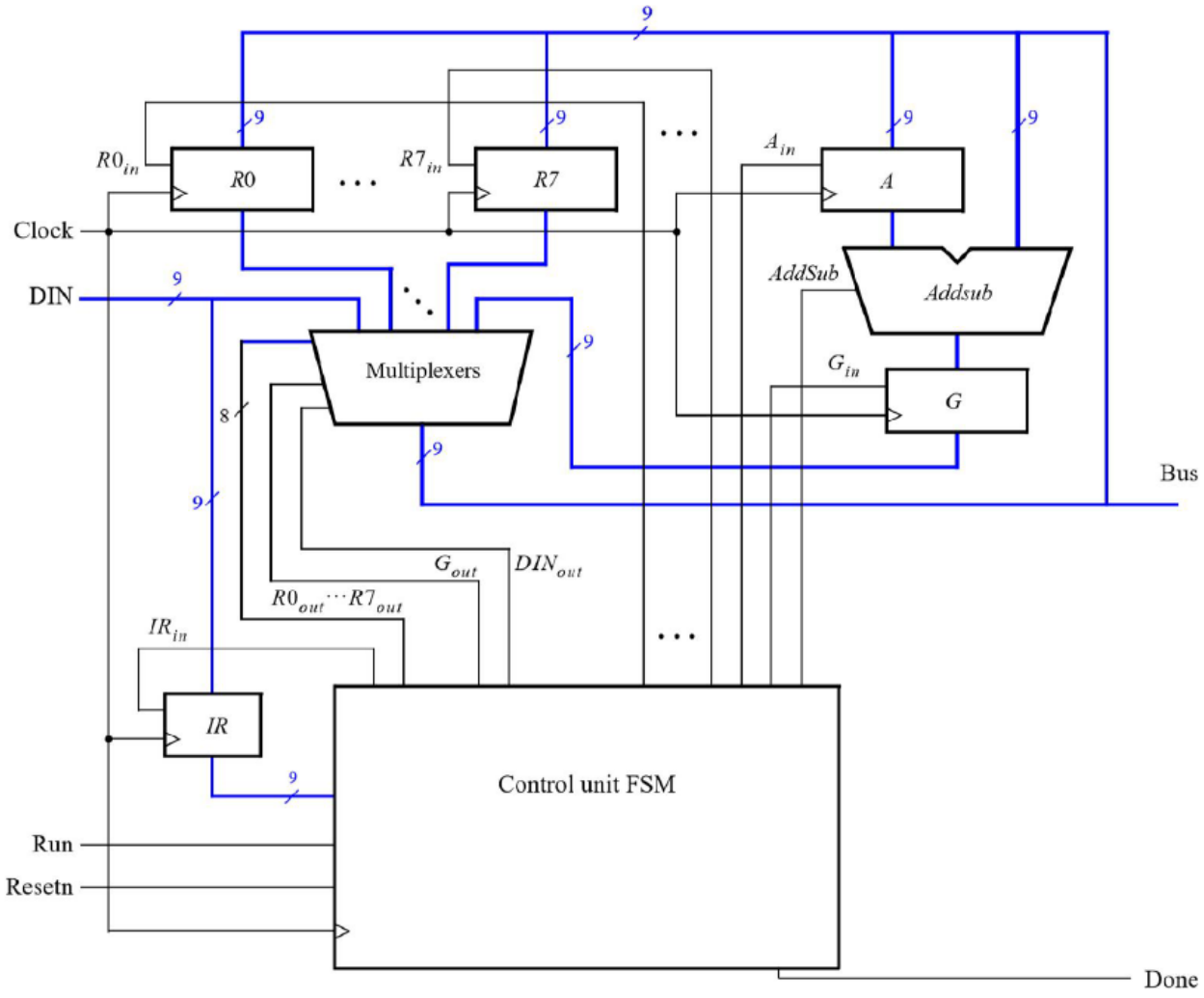
18/07/2020
308304476 דוד קאופמן
305628125 יואב זילברשטיין

מעבדה ספרתית 83394

דו"ח 3 - A Simple Processor

במעבדה זו מימשנו מעבד 9 ביט Multi-cycle בסיסי, המבצע פקודות המתקבלות בכניסה DIN.

<u>סכמה כללית של המעבד</u>



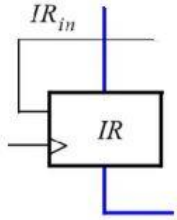
מבנה המעבד

המעבד מחולק לבלוקים המבצעים תפקיד שונה ומחוברים ביניהם.

<u>זיכרון</u>

במעבד קיימים 11 רגיסטרים בגודל 9 ביט כל אחד $(R0 - R7, A, G, IR)$. הרגיסטרים הם סינכרוניים ופועלים בעליית שעון.

רגיסטר IR



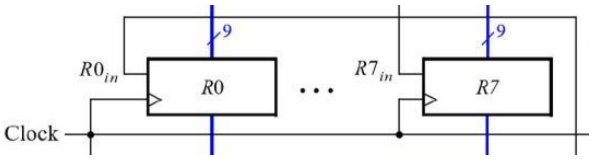
שומר את הפקודה הנוכחית שיש לבצע. מקבל את הפקודה מהכניסה D_{in}.

<u>כניסות</u>

- clk - אות שעון.
- D_{in} - באס של 9 ביטים המכילים את הפקודה.
- IR_{in} - ביט אפשור כתיבה לרגיסטר.

<u>יציאה</u>: (9 ביט) מחוברת ליחידת הבקרה לצורך פיענוח וביצוע הפקודה.

רגיסטרים $R0 - R7$



רגיסטרים לשימוש כללי. שומרים את המידע עליו מתבצעות הפעולות השונות של המעבד.

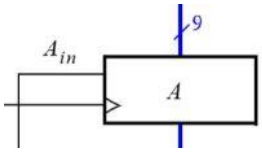
<u>כניסות</u>

- clk - אות שעון.
- $BusWires$ - באס של 9 ביטים עליו נמצא המידע שאותו אנחנו רוצים לכתוב לרגיסטר.
- R_{in} - אפשור כתיבה לרגיסטר.

<u>יציאה</u>

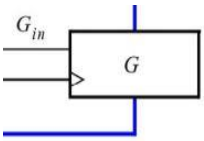
מחוברת למרבב בכדי להעביר את הערך השמור הלאה.

רגיסטרים A, G



מיועדים לביצוע פעולות האריתמטיות. רגיסטר A שומר ערך המיועד לפעולה האריתמטית, רגיסטר G דוגם ושומר את התוצאה של הפעולה האריתמטית.

<u>כניסות</u>



- clk - אות שעון.
- $BusWires$ - באס של 9 ביטים עליו נמצא המידע אותו אנחנו רוצים לכתוב לרגיסטר.
- A_{in}, G_{in} - אפשור כתיבה לרגיסטר A, G בהתאמה.

<u>יציאה</u>

היציאה של A מחוברת ל ALU בכדי שנוכל לבצע את הפעולה האריתמטית. היציאה של G מחוברת למרבב בכדי שנוכל להעביר ולשמור את התוצאה באחד מהרגיסטרים $R0 - R7$.

<u>ALU - Arithmetic Logic Unit</u>

יחידה אסינכרונית שאינה תלויה באות שעון האחראית על ביצוע פעולות אריתמטיות ולוגיות. במעבד שלנו נדרשת לבצע רק פעולות של חיבור וחיסור בהתאם לאות בקרה. משום שהכניסות הן 9 ביט וגם היציאה היא 9 ביט יש לשים לב שכאשר אנחנו מבצעים פעולות של חיבור או חיסור יתכן שנקבל מצב של Overflow כלומר, התוצאה מכילה ביט נוסף. לאחר התייעצות עם גבי נאמר לנו שאין צורך להתחשב במצב כזה.

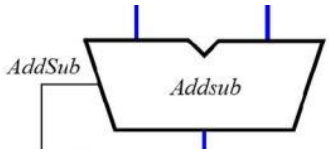
כניסות

- A - 9 ביטים ששמורים ברגיסטר A שהם הערך הראשון.
- B - 9 ביטים ששמורים על ה $BusWires$ שהם הערך השני.
- $SelectOP$ – אות בקרה של ביט אחד. ב-0 לוגי ביצוע חיסור וב-1 לוגי ביצוע חיבור.

יציאה $ALUout$

התוצאה של הפעולה האריתמטית. 9 ביטים שמועברים לרגיסטר G לצורך שמירה.

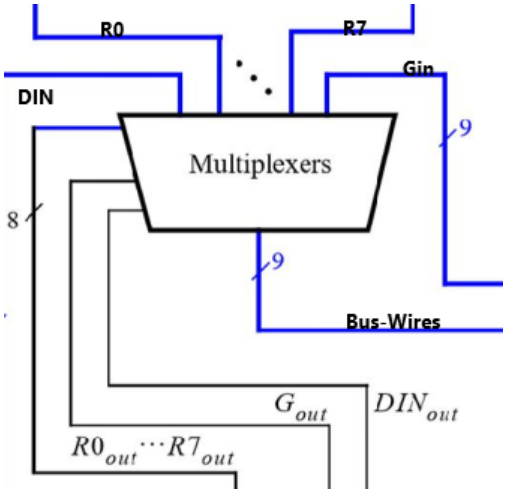
מימוש הרכיב בקוד




```
module addSub(A, B, selectOP, ALUout);
    input wire [8:0]A, B;
    input wire selectOP;

    output wire [8:0]ALUout;

    parameter ADD = 1'b1,
              SUB = 1'b0;

    assign ALUout = (selectOP == ADD)? A + B: A - B;

endmodule
```

מרבב Multiplexer:

יחידה אסינכרונית שאינה תלויה באות שעון האחראית על ניתוב המידע בין הערוצים השונים במעבד ע"פ אותות הבקרה שהוא מקבל. לדוגמא, העברת ערך מרגיסטר שאותו אנחנו רוצים לקרוא אל ה-$BusWires$ בכדי לבצע פעולה אריתמטית על הערך.



כניסות

- $R0 - R7$ - 9 כניסות, כל כניסה היא באס של 9 ביטים המגיע מהרגיסטר המתאים
- G_{in} - באס של 9 ביטים המגיע מרגיסטר G.
- D_{in} - באס של 9 ביטים המאפשר לנו להעביר את הכניסה D_{in} ישירות אל ה-$BusWires$. משתמשים באופציה זו כאשר רוצים לבצע פעולה על ערך מספרי $Immediate$.
- SR_{out} - אות בקרה. באס של 9 ביטים שאומר איזה מוצא, מבין המוצאים של הרגיסטרים $R0 - R7$, יש להעביר את ל-$BusWires$.
- SG_{out} - אות בקרה. ביט אחד שאומר לנו האם להעביר את המוצא של הרגיסטר G ל-$BusWires$.
- SD_{out} - אות בקרה. ביט אחד שאומר לנו האם להעביר את הכניסה D_{in} ישירות ל-$BusWires$.

יציאה $Muxout$

העברת המידע ל-$BusWires$.

מוצא המרבב כתלות באותות הבקרה:

מוצא המרבב	אותות הבקרה		
Mux_{out}	SR_{out}	SG_{out}	SD_{out}
$R0$	00000001	0	0
$R1$	00000010	0	0
$R2$	00000100	0	0
$R3$	00001000	0	0
$R4$	00010000	0	0
$R5$	00100000	0	0
$R6$	01000000	0	0
$R7$	10000000	0	0
D_{in}	00000000	0	1
G_{in}	00000000	1	0

מימוש הרכיב בקוד

```
module multiplexers(R0, R1, R2, R3, R4, R5, R6, R7, Gin, Din, SRout, SGout, SDout, MUXout);

    input wire  [8:0]R0, R1, R2, R3, R4, R5, R6, R7;

    input wire  [8:0]Gin;
    input wire  [8:0]Din;           // 9 bit
    input wire [7:0]SRout;          // 8 bit
    input wire SGout, SDout;        // 1 bit

    output wire [8:0] MUXout;

    parameter zero = 8'b0;
    assign MUXout = (SGout == 1)     ? Gin:
                    (SDout == 1)     ? Din:
                    (SRout == 8'h1) ? R0:   //00000001
                    (SRout == 8'h2) ? R1:   //00000010
                    (SRout == 8'h4) ? R2:   //00000100
                    (SRout == 8'h8) ? R3:   //00001000
                    (SRout == 8'h10)? R4:   //00010000
                    (SRout == 8'h20)? R5:   //00100000
                    (SRout == 8'h40)? R6:   //01000000
                    (SRout == 8'h80)? R7:   //10000000
                   R0;                      //default
endmodule
```

יחידת הבקרה Control Unit FSM

יחידה סינכרונית שפועלת בעליית אות השעון. יחידה זו ממומשת כמכונת מצבים כך שבכל מצב יוצאים אותות הבקרה הרלוונטיים לפקודה ולשלב הנוכחי שלה. מכונת המצבים תתואר בהמשך.

כניסות

clk - אות שעון.

$Resetn$ - אות ריסט.

Run - המעבד מתחיל לבצע את הפקודה כאשר סיגנל זה עולה ל 1 לוגי.

IR - המוצא של רגיסטר IR מוכנס לתוך יחידת הבקרה בכדי שנוכל לפענח את הפקודה שיש לבצע.

<u>יציאות:</u>

היציאות הם אותות הבקרה.

<u>אותות בקרה עבור כתיבה לרגיסטר:</u>

- IR_{in} – ביט יחיד המאפשר כתיבה לרגיסטר IR ב 1 לוגי.
- G_{in} – ביט יחיד המאפשר כתיבה לרגיסטר G ב 1 לוגי.
- A_{in} – ביט יחיד המאפשר כתיבה לרגיסטר A ב 1 לוגי.
- $R0_{in} - R7_{in}$ - באס של 8 ביטים. כל ביט מאפשר כתיבה לרגיסטר המתאים ב 1 לוגי. לדוגמא, עבור הערך 00000001 תתאפשר כתיבה לרגיסטר $R0$. בצורה זו ניתן לאפשר כתיבה למספר רגיסטרים ביחד.

<u>אותות בקרה עבור קריאה מרגיסטר</u>

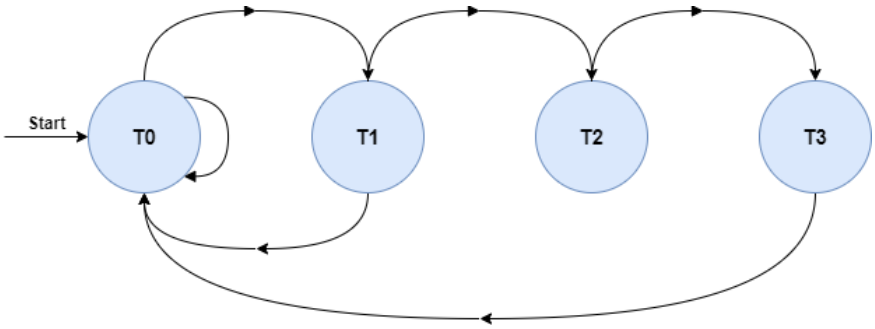
- DIN_{out} – ביט יחיד שאומר למרבב שעליו להעביר את הכניסה DIN ישירות ל $BusWires$.
- G_{out} – ביט יחיד שאומר למרבב שעליו להעביר את הערך שנמצא ברגיסטר G ל $BusWires$.
- $R0_{out} - R7_{out}$ – באס של 8 ביטים שאומר למרבב להעביר את הערך של רגיסטר מסוים ל-$BusWires$. לדוגמא, עבור הערך 00000001 הערך שברגיסטר $R0$ יועבר ל-$BusWires$.

<u>אותות בקרה נוספים</u>

- $AddSub$ - ביט יחיד שאומר לרכיב $AddSub$ איזה פעולה עליו לבצע. הגדרנו שעבור 0 לוגי יתבצע חיסור ועבור 1 לוגי יתבצע חיבור.
- $Done$ - משמש כדגל שאומר שביצוע הפקודה הסתיים.

<u>מכונות המצבים</u>

ליבו של המעבד ממומש כמכונת מצבים שמחליפה מצב בעת עליית שעון. בכל מצב מתבצע חלק מסוים של הפקודה. נשים לב שיש פקודות שצורכות פחות שלבים לצורך ביצוען. בהתאם לשלב הנוכחי של הפקודה, יוצאים אותות בקרה שונים.



כאשר מתקבל אות RUN (ובעלית שעון) המעבד מתחיל לבצע פקודה חדשה, מצב T0. במצב זה הפקודה החדשה נשמרת ברגיסטר IRin, ופקודה מפוענחת.

בעליית אות שעון המוכנה עוברת למצב T1. אם מדובר בפקודות mv או mvi אז באות השעון הבא יתקבל Done ואנחנו נחזור למצב T0.

בשאר הפקודות, בעליית השעון המכונה תעבור למצב T2, ולאחר עליית שעון נוספת תעבור למצב T3 - המצב האחרון. בסופו יתקבל אות Done ונחזור למצב T0.

בבדיקת הכרטיס על גבי הכרטיס, הפקודות והערכים מוזנים למעבד באופן ידני (בעזרת המתגים שעל הכרטיס) לכן לא נוכל להשתמש בשעונים של $27MHz$ או $50MHz$ כי הם מהירים מדי. משום כך, אות השעון יהיה לחיצה על כפתור Key1 שעל הלוח.

אותות הבקרה שיוצאים מיחידת הבקרה בכל מצב:

Operation	To	T1	T2	T3
mv	IR_{in}	$RY_{out}, RX_{in}, Done$		
mvi	IR_{in}	$DIN_{out}, RX_{in}, Done$		
add	IR_{in}	RX_{out}, A_{in}	$RY_{out}, AddSub = 1, G_{in}$	$G_{out}, RX_{in}, Done$
sub	IR_{in}	RX_{out}, A_{in}	$RY_{out}, AddSub = 0, G_{in}$	$G_{out}, RX_{in}, Done$
$addi$	IR_{in}	RX_{out}, A_{in}	$DIN_{out}, AddSub = 1, G_{in}$	$G_{out}, RX_{in}, Done$
$mviAll$	IR_{in}	$DIN_{out}, RX_{in} = 8'hFF, Done$		

חשוב לשים לב כי הטבלה מתארת את אותות הבקרה שעלינו לקבוע ל1 לוגי בכל שלב. יש לדאוג שאותות בקרה שבהם לא משתמשים בשלב הנוכחי יהיו קבועים ל0 לוגי.

מבנה של פקודה

כל פקודה מיוצגת ע"י 9 ביטים באופן הבא: $III\ XXX\ YYY$

III – 3 ביטים שמייצגים את הפקודה שיש לבצע. הפקודות:

Code	Operation	Function Preformed
000	$mv\ R_x, R_y$	$R_x \leftarrow [R_y]$
001	$mvi\ R_x, \#D$	$R_x \leftarrow D$
010	$add\ R_x, R_y$	$R_x \leftarrow R_x + [R_y]$
011	$sub\ R_x, R_y$	$R_x \leftarrow R_x - [R_y]$
100	$addi\ R_x, \#D$	$R_x \leftarrow R_x + D$
101	$mviall$	$R_0, R_1 \cdots R_7 \leftarrow D$

XXX – 3 ביטים שמייצגים את רגיסטר היעד R_x. ברגיסטר זה תשמר התוצאה הסופית של הפקודה.

YYY – 3 ביטים שמייצגים רגיסטר נוסף, R_y, שאיתו נבצע את הפקודה. נציין שבפקודות כמו mvi או $addi$, בהן עושים פעולה עם מספר *(Immediate)*, אנחנו מתעלמים ממה שמופיע בשלוש הביטים הללו.
פרוט על אופן הביצוע של הפקודות עצמן יבוא בהמשך.

<u>פקודות</u>

הקוד של הפקודות מצורף בהמשך

בכל הפקודות מבוצע השלב הראשון **T0**- קריאה ופענוח של הפקודה, רגיסטר IR מאופשר דרך כניסת IR_{in}.

באופן זה מוזנת לרגיסטר IR הפקודה שמגיעה בכניסות DIN.

פקודת MV

פקודה זו מעתיקה את התוכן של רגיסטר אחד לשני.

T0

T1:

- RY_{out} – קריאה מרגיסטר. אומר למולטיפלקסר להעביר את הערך שברגיסטר ל $BusWires$.
- RX_{in} – מעלים ל 1 לוגי כדי לאפשר כתיבה לרגיסטר – הערך שנמצא על ה $BusWires$ נכתב אל הרגיסטר.
- $Done$ - מעלים ל 1 לוגי שאומר שסיימנו את הפקודה וניתן לעבור לפקודה הבאה.

פקודת MVI

פקודה זו מעבירה ערך מספרי מהכניסה DIN לתוך רגיסטר.

T0

T1:

- DIN_{out} - קוראים את הערך המספרי שמופיע בכניסה DIN. אומר למולטיפלקסר שיש להעביר את ערך הכניסה DIN ל $BusWires$.
- RX_{in} - מעלים ל 1 לוגי כדי לאפשר כתיבה לרגיסטר – הערך שנמצא על ה $BusWires$ נכתב אל הרגיסטר.
- $Done$ - מעלים ל 1 לוגי שאומר שסיימנו את הפקודה וניתן לעבור לפקודה הבאה.

פקודת ADD

פקודה זו מבצעת חיבור בין שני רגיסטרים ושומרת את התוצאה ברגיסטר היעד.

T0

T1:

- RX_{out} - קריאה מרגיסטר. אומר למולטיפלקסר להעביר את הערך שברגיסטר ל $BusWires$.

- A_{in} - מעלים ל 1 לוגי כדי לאפשר כתיבה לרגיסטר A – הערך שנמצא על ה $BusWires$ נכתב אל רגיסטר A.

T2:

- RY_{out} - קריאה מרגיסטר. אומר למולטיפלקסר להעביר את הערך שברגיסטר ל $BusWires$.
- $AddSub = 1$ – אומרים ל ALU לבצע פעולת חיבור (קבענו ש 1 לוגי הוא ביצוע פעולת חיבור).
  ה ALU מחבר את הערך שנמצא ברגיסטר A והערך שנמצא על ה $BusWires$.
- G_{in} – מעלים ל 1 לוגי כדי לאפשר כתיבה לרגיסטר G. התוצאה של פעולת החיבור (מוצא ה ALU) נכתבת אל רגיסטר G.

T3:

- G_{out} -קוראים את הערך שנמצא ברגיסטר G כלומר, אומרים למולטיפלקסר להעביר את הערך שברגיסטר G ל $BusWires$.
- RX_{in} – מעלים ל 1 לוגי כדי לאפשר כתיבה לרגיסטר – הערך שנמצא על ה $BusWires$ נכתב אל הרגיסטר.
- $Done$ – מעלים ל 1 לוגי שאומר שסיימנו את הפקודה וניתן לעבור לפקודה הבאה.

פקודת SUB

פקודה זו מבצעת חיסור בין שני רגיסטרים ושומרת את התוצאה ברגיסטר היעד.

T0

T1:

- RX_{out} – קריאה מרגיסטר. אומר למולטיפלקסר להעביר את הערך שברגיסטר ל $BusWires$.
- A_{in} – מעלים ל 1 לוגי כדי לאפשר כתיבה לרגיסטר A – הערך שנמצא על ה $BusWires$ נכתב אל רגיסטר A.

T2:

- RY_{out} - קריאה מרגיסטר. אומר למולטיפלקסר להעביר את הערך שברגיסטר ל $BusWires$.
- $AddSub = 0$ – אומרים ל ALU לבצע פעולת חיסור (קבענו ש 0 לוגי הוא ביצוע פעולת חיסור).
  ה ALU מחסר את הערך שנמצא ברגיסטר A והערך שנמצא על ה $BusWires$.
- G_{in} – מעלים ל 1 לוגי כדי לאפשר כתיבה לרגיסטר G. התוצאה של פעולת החיסור (מוצא ה ALU) נכתבת אל רגיסטר G.

T3:

- G_{out} -קוראים את הערך שנמצא ברגיסטר G כלומר, אומרים למולטיפלקסר להעביר את הערך שברגיסטר G ל $BusWires$.

- RX_{in} – מעלים ל 1 לוגי כדי לאפשר כתיבה לרגיסטר – הערך שנמצא על ה $BusWires$ נכתב אל הרגיסטר.
- $Done$ – מעלים ל 1 לוגי שאומר שסיימנו את הפקודה וניתן לעבור לפקודה הבאה.

פקודת ADDI

פקודה זו מאפשרת לבצע חיבור בין רגיסטר לערך מספרי שנמצא בכניסה DIN. התוצאה נשמרת אל רגיסטר היעד.

T0

T1:

- RX_{out} – קריאה מרגיסטר. אומר למולטיפלקסר להעביר את הערך שברגיסטר ל $BusWires$.
- A_{in} – מעלים ל 1 לוגי כדי לאפשר כתיבה לרגיסטר A – הערך שנמצא על ה $BusWires$ נכתב אל רגיסטר A.

T2:

- DIN_{out} – קוראים את הערך המספרי שמופיע בכניסה DIN. אומר למולטיפלקסר שיש להעביר את ערך הכניסה DIN ל $BusWires$.
- $AddSub = 1$ – אומרים ל ALU לבצע פעולת חיבור (קבענו ש 1 לוגי הוא ביצוע פעולת חיבור).
  ה ALU מחבר את הערך שנמצא ברגיסטר A והערך שנמצא על ה $BusWires$.
- G_{in} – מעלים ל 1 לוגי כדי לאפשר כתיבה לרגיסטר G. התוצאה של פעולת החיסור (מוצא ה ALU) נכתבת אל רגיסטר G.

T3:

- G_{out} -קוראים את הערך שנמצא ברגיסטר G כלומר, אומרים למולטיפלקסר להעביר את הערך שברגיסטר G ל $BusWires$.
- RX_{in} – מעלים ל 1 לוגי כדי לאפשר כתיבה לרגיסטר – הערך שנמצא על ה $BusWires$ נכתב אל הרגיסטר.
- $Done$ – מעלים ל 1 לוגי שאומר שסיימנו את הפקודה וניתן לעבור לפקודה הבאה.

פקודת MAVIALL

פקודה זו מאפשרת לכתוב ערך מספרי מהכניסה DIN אל כל הרגיסטרים. יכול להיות שימושי כאשר אנחנו רוצים לאפס את כל הרגיסטרים יחד (לנקות את הזיכרון) במקום לכתוב אפס לכל רגיסטר בנפרד.

T0

T1:

- DIN_{out} – קוראים את הערך המספרי שמופיע בכניסה DIN. אומר למולטיפלקסר שיש להעביר את ערך הכניסה DIN ל $BusWires$.
- RX_{in} - בכדי לכתוב לכל הרגיסטרים יחד אנחנו צריכים לאפשר את כולם ולכן, אנחנו נותנים את הערך $8'FF = 11111111$ כלומר 8 אחדות – הערך שנמצא על ה $BusWires$ נכתב אל **כל** הרגיסטרים יחד.
- $Done$ – מעלים ל 1 לוגי שאומר שסיימנו את הפקודה וניתן לעבור לפקודה הבאה.

הקונטרולים ששולטים על המולטיפלקסר מוגדרים באופן הבא:

- הבאס של 8 ביטים $R0_{out} - R7_{out}$ מוגדר בקוד $SRout$
- G_{out} בקוד נקרא $SGout$
- DIN_{out} נקרא אצלנו בקוד $SDout$

קטע קוד בו ניתן לראות את הקונטרולים בכל שלב של פקודה (מתוך קובץ proc.v):

```
1       // Control FSM outputs
2       always @(Tstep_Q or I or Xreg or Yreg) begin
3
4           //... specify initial values
5           Ain <= 1'b0;
6           Gin <= 1'b0;
7           Done <= 1'b0;
8           Rin <= 8'b0;
9           SDout <= 1'b0;
10          SGout <= 1'b0;
11          SRout <= 8'b0;
12          IRin <= 1'b0;
13
14          case (Tstep_Q)
15              T0: begin                   // Signals in Time Step 0
16                      IRin <= 1'b1;       // Enable write to IR register. Store DIN in IR
17                  end
18
19              T1: begin                   // Signals in Time Step 1
20                      case (I)
21                          MV: begin
22                              Rin  <= Xreg;   // enable write to register
23                              SRout <= Yreg;  // select register to read
24                              Done <= 1'b1;
25                              end
26
27                          MVI: begin
28                              SDout <= 1'b1;  // read immidate number from DIN
29                              Rin  <= Xreg;   // enable write to register
30                              Done <= 1'b1;
31                              end
32
33                          ADD: begin
34                              SRout <= Xreg;  // read from register
35                              Ain <= 1'b1;    // enable write to register A
36                              end
37
38                          SUB:begin
39                              SRout <= Xreg;  // read from register
40                              Ain <= 1'b1;    // enable write to register A
41                              end
42
43                          ADDI: begin
44                              SRout <= Xreg;  // read from register
45                              Ain <= 1'b1;    // enable write to register A
46                              end
47
48                          MVIALL: begin
49                              SDout <= 1'b1;  // read immidate number from DIN
50                              Rin  <= 8'hFF;  // enable write to all registers
51                              Done <= 1'b1;
52                              end
53
54                          default:
55                              Done <= 1'b1;
56
57                      endcase //case(I)
58                  end //T1
59
60              T2: begin           // Signals in Time Step 2
61
62                      Gin <= 1'b1;                // enable write to register G
63
64                      case(I)
65                          ADD:begin
66                              SRout <= Yreg;      // read from register
67                              selectOP <= 1'b1;   // tell ALU to do ADD
68                              end
69
70                          SUB:begin
71                              SRout <= Yreg;      // read from register
72                              selectOP <= 1'b0;   // tell ALU to do SUB
73                              end
```

בכל מחזור מבוצע איפוס של כל הקונטרולים.
איפוס הקונטרונים מונע פגיעה בפעולת
המעבד, רק הקונטרולים הרלוונטיים ידלקו
בכל מחזור.

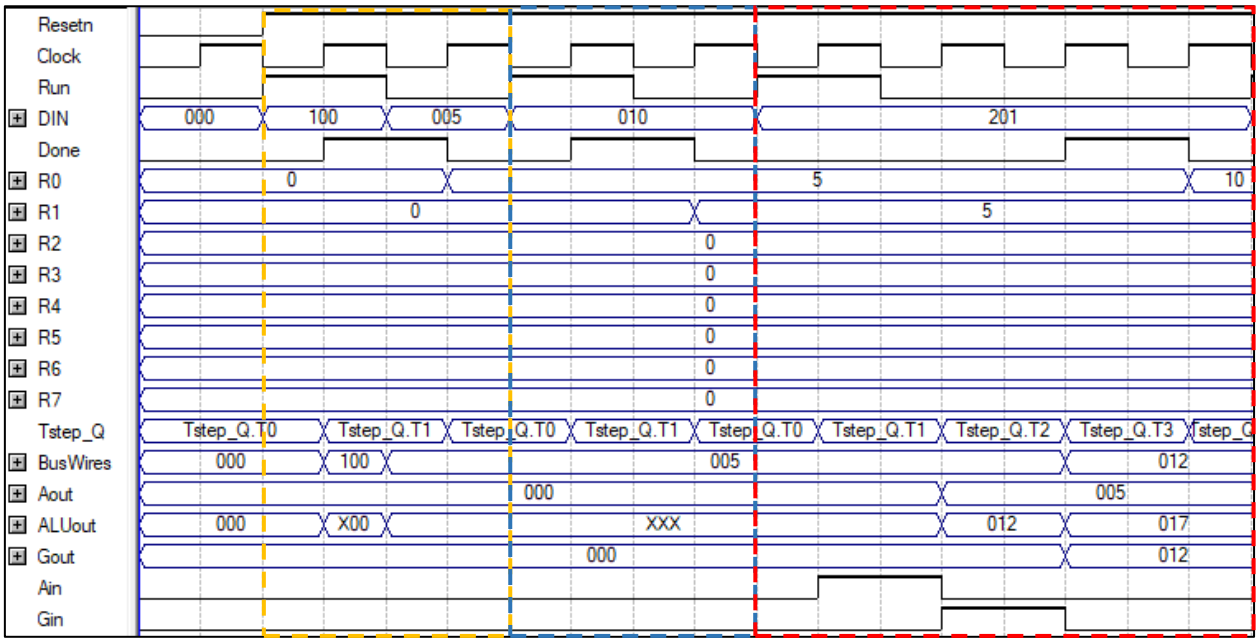
```
                        ADDI:begin
                            SDout <= 1'b1;      // read immidate number from DIN
                            selectOP <= 1'b1;   // tell ALU to do ADD
                            end
                    endcase //case(I)
                end //T2


            T3: begin       // Signals in Time Step 2

                SGout <= 1'b1;  // read from register G
                Rin <= Xreg;    // enable write to register
                Done <= 1'b1;

                end //T3
        endcase //case(Tstep_Q)

    end //always
```

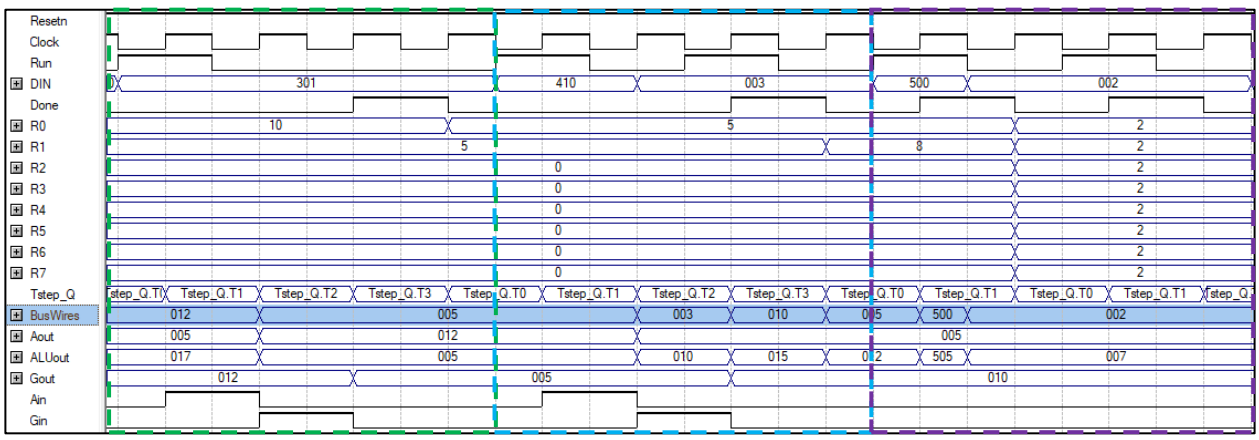
הקוד המלא של proc.v מצורף.

סימולציות

<u>בדיקות בסיסיות</u>

<u>סימולציית Functional</u>



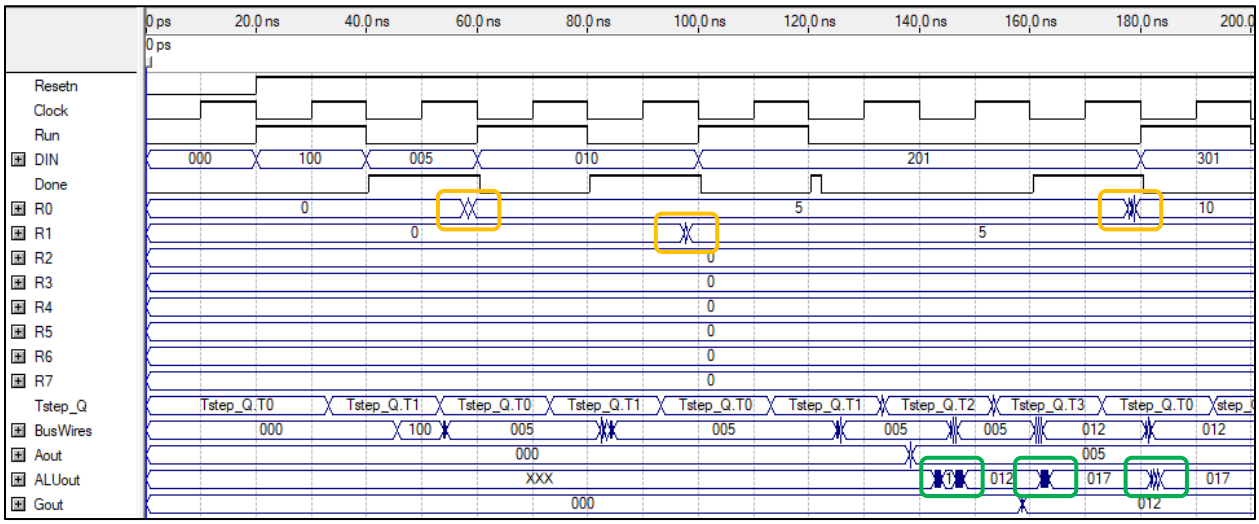


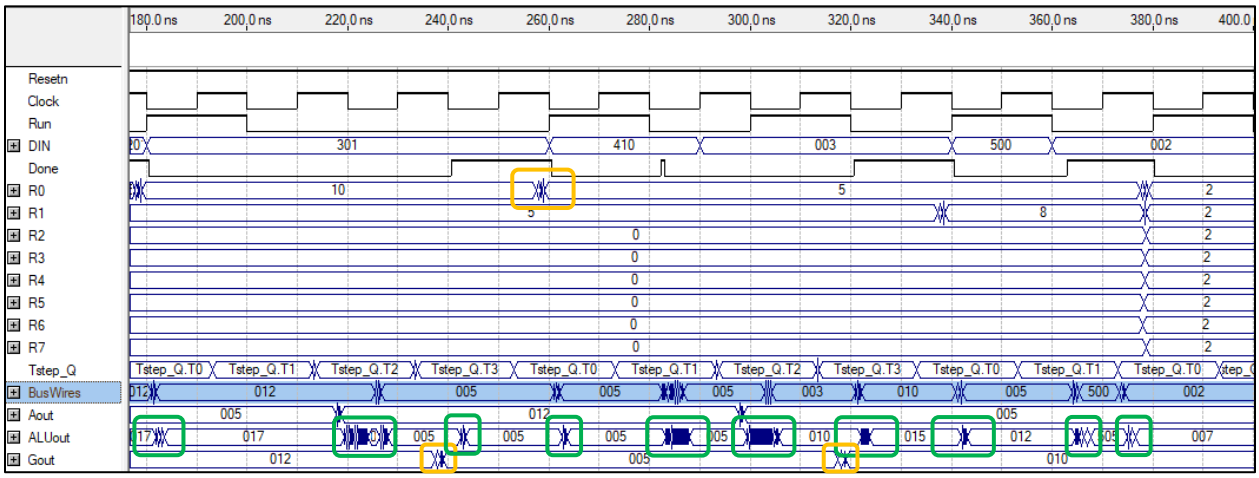
- $5 \rightarrow R_0$: MVI
- $R_0 - R_1 \rightarrow R_0$: SUB
- $R_0 \rightarrow R_1$: MV
- $R_1 + 3 \rightarrow R_1$: ADDI
- $R_0 + R_1 \rightarrow R_0$: ADD
- $2 \rightarrow R_0 - R_7$: MVIALL

מהסימולציה ניתן לראות שהפקודות עובדת כנדרש.

הסימולציה אינה עוברת על כל המצבים ועל מקרה קיצון, בהמשך נתייחס לכך.

סימולציית Timing



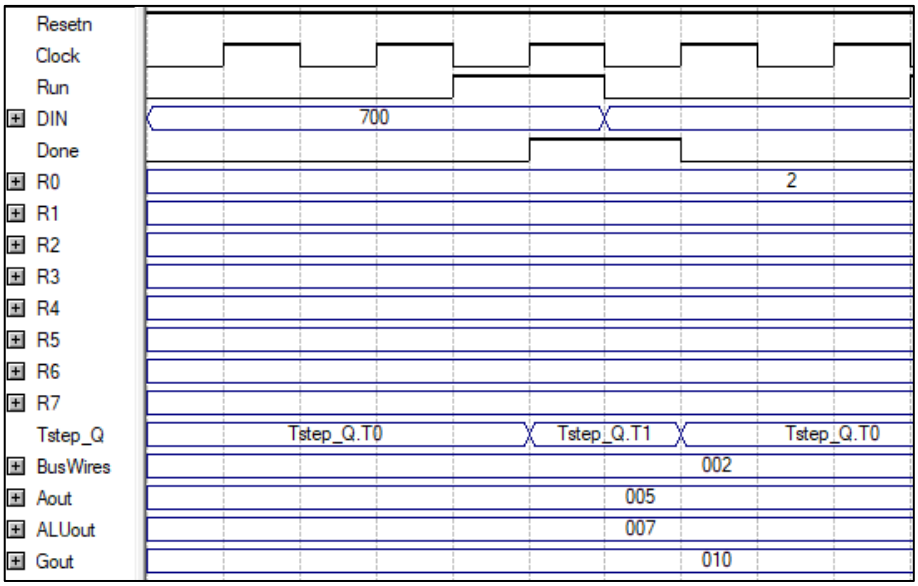


- קפיצות במעברים הנובעות ממעבר של ביטים מדורג בbus, הנגרם ממגבלה חומרתית. משך זמן המיתוג בין בית לביט הוא שונה בגלל שוני חומרתי שעשוי להיות מרמת הטרנזיסטור.
- ALU רכיב אסינכרוני, לכן המוצא משתנה כל הזמן בהתאם לשינויים בכניסות. לכן רגיסטר G המחובר למוצא הALU חשוב, הוא דוגם את הערך במוצא בתזמון הנכון, ומאפסן בו את הערך.

מהסימולציות ניתן לראות שהלוגיקה תקינה, אין בעיות תזמונים (בבדיקות הנ"ל).

בדיקות נוספות

1. בדיקת פקודה לא מוגדרת

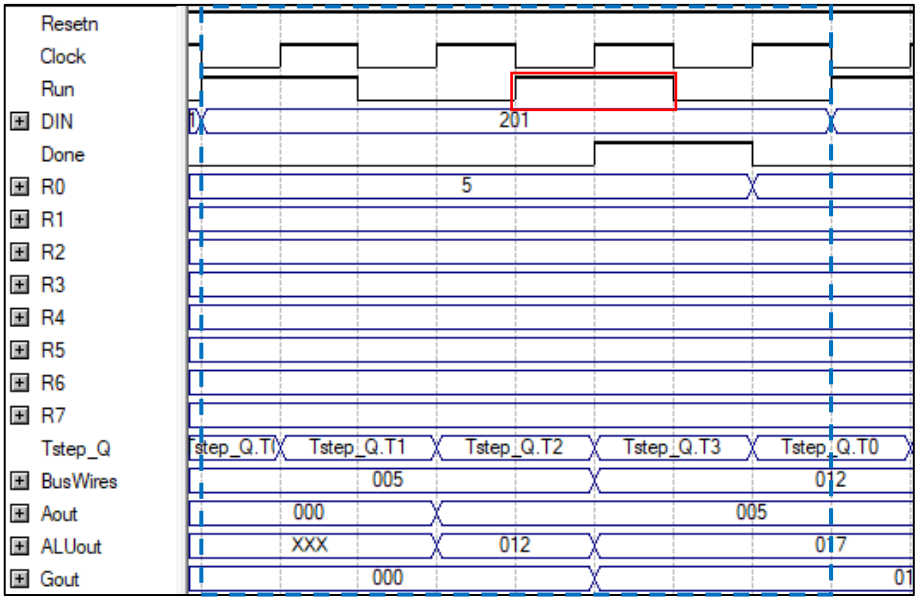


בסימולציה (functional) הגדרנו פקודה לא מוגדרת (לא אחת מבין שש הפקודות המוגדרות).

עבור פקודה לא מוגדרת הגדרנו בשלב T1 העלאת דגל Done=1'b1 המציין סיום פקודה, שאר הערכים במעבד נשארים ללא שינוי.

בנוסף ניתן לראות שהפקודה נדגמת בעליית שעון ומותנת בלחיצה על כפתור run.

2. לחיצה על run בזמן ריצה של פקודה



בסימולציה ניתן לראות ריצה של פוקדת ADD, במהלך הריצה הוספנה לחיצה של run.

ניתן לראות שהלחיצה לא פוגעת בריצת הפקודה.

מפלט הקומפלציה מתקבל שהתדר המקסימלי בו המעבד יכול לעבוד הוא 79.96 MHz.

הבדיקות שביצענו אינן מכסות את כל המקרים האפשריים.
נושא הבדיקות הוא אתגר רציני. בכיתה דנו בכך, הוסבר לנו בכלליות על נוהל בדיקות המקובל באינטל, כך שעל כל כותב קוד יש כ- 1.5 אנשים שבודקים את הקוד, ובנוסף נהוג לבצע מידול של המעבד/ מודול/ רכיב בשפת תכנות ולהכניס קלטים שונים (מושכלים) ולהשוואות פלטים במטרה לכסות את מרבית המקרים.

סיכום

בדו"ח מימשנו מעבד Multi-cycle, כל פקודה מבוצעת במספר שונה של מחזורי שעון, ובכך זמן המחזור של פקודת מסוימת יתקצר, כיוון שזמן המחזור של כל פקודה לא נקבע על ידי זמן ביצוע הפקודה הארוכה ביותר כמו בsingle-cycle. בנוסף נוכל לעבוד עם תדר עבודה מהיר יותר משל הsingle-cycle.

יתרונות:

- כל פקודה במספר שונה של מחזורי שעון, ובכך זמן ביצוע הפקודות נ
- אין כפילות חומרה

חסרונות:

- מצריך שימוש במספר גדול יותר של רגיסטרים בהשוואה לSingle-cycle, כדי לשמור את הערכי הביניים במהלך ביצוע הפקודה.

על מנת להפוך את המעבד לsingle-cycle, דרוש שכל הפקודות יבוצעו במחזור שעון אחד שיקבע ע"פ זמן ביצוע הפקודה הארוכה ביותר (זמן המחזור יהיה לכל הפחות הזמן מחזור של הפקודה הארוכה ביותר).

קישור לסרטון הרצת מספר פקודות על הכרטיס FPGA
https://youtu.be/9EkBlEJSea0