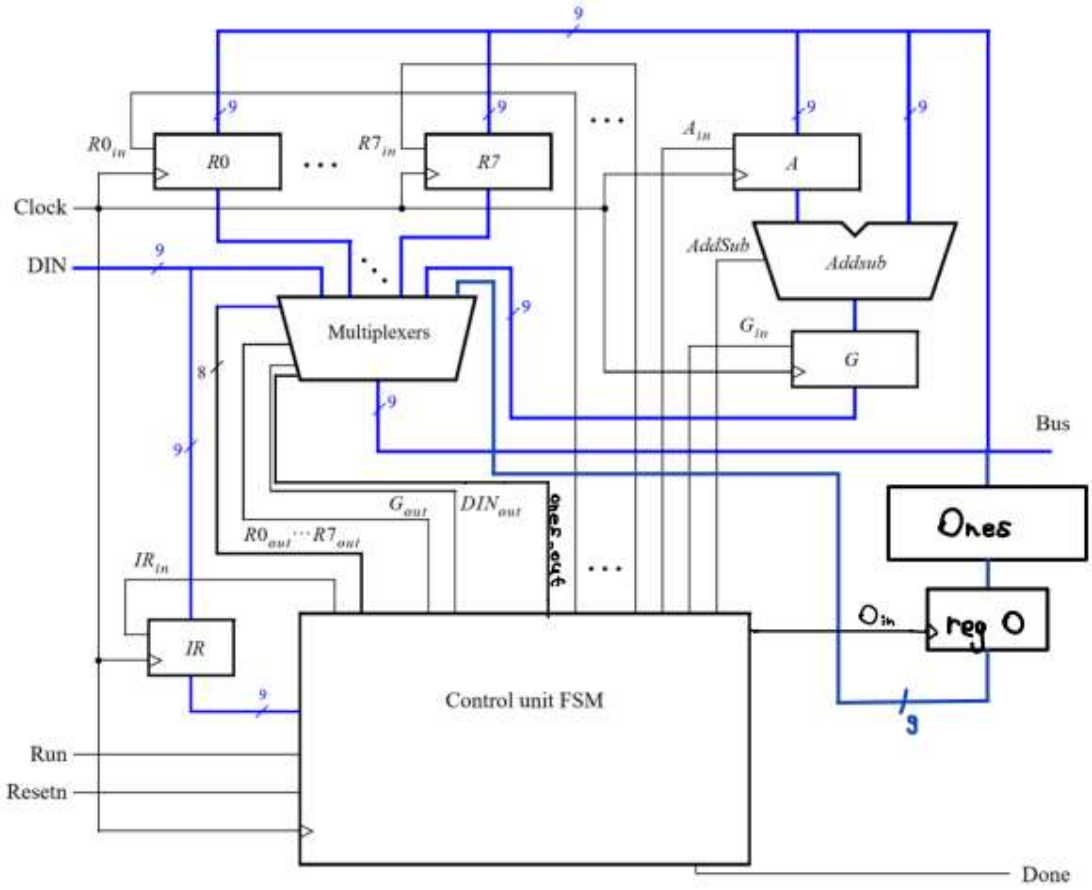
בס״ד

Simple Processor



מגישים : שובי פסקו

יוסף סוקוליק

במעבדה זו נדרשנו להשתלב במימוש של מעבד פשוט כלומר, קיבלנו skeleton של קוד ורילוג והיינו צריכים להבין את חלקי הקוד הקיים ולהשלים את החלקים החסרים.

המעבד שנדרשנו לממש אמור לבצע מס' פקודות אריתמטיות ופקודות זיכרון, נפרט את הפקודות השונות:

- פקודת mvi- פקודה בה אנו כותבים ערך מספרי לזיכרון.
- פקודת mv- פקודה בה אנו מעתיקים ערך הכתוב בתא אחד בזיכרון לתא שני.
- פקודת add- פקודת חיבור בין שני מספרים השמורים בזיכרון.
- פקודת sub- פקודת חיסור בין שני מספרים השמורים בזיכרון.

בנוסף, נדרשנו לממש פקודה נוספת אותה המעבד יוכל לבצע

- פקודת num of ones- פקודה המונה את מס' האחדות מהן מורכב מס' השמור בזיכרון ומכניסה אותה לתא אחר בזיכרון.

למעבד ישנו mux המנהל את הזיכרון (קריאה/ כתיבה), זיכרון של שמונה רגיסטרים בני 9-bit כל אחד, יחידת ALU המבצעת את הפעולות האריתמטיות ויחידה המבצעת את ספירת האחדות.

המצבים במכונת המצבים המממשת את המעבד הם מחזורי השעון הדרושים לביצוע כל פקודה. ישנן פקודות הדורשות שני מחזורי שעון בלבד למילואן וישנן פקודות הדורשות ארבעה מחזורי שעון. כאשר פקודה אחת מתבצעת הפקודה האחרת ממתינה ואינה מתחילה להתבצע. הארכיטקטורה של מעבד זה דומה לארכיטקטורת Multi-Cycle-MIPS שלמדנו בקורס "מבנה המחשב".

נפרט מה מתבצע בכל מחזור שעון בפקודות השונות. כאשר בכל מחזור שעון אנחנו מעלים את הסיגנלים הנצרכים.

בכל הפקודות המחזור הראשון זהה:

T_0—המעבד מקבל את הפקודה ואת הרגיסטרים מ- Din ע"י המשתמש דרך הSW שבכרטיס המשתמש מכניס 9 ביטים שהם הפקודה (I) ומיקום הרגיסטרים R_X וR_Y (זה מסודר כך: IIIXXXYYY). המעבד שומר את הפקודה ומיקומי הרגיסטרים ברגיסטר IR.

- עבור פקודת mvi, הכנסת מילה (בגודל 9-bit) מDin לזיכרון, (opcode 000) $R_x <= Int$.

T_1 – המעבד בורר את המילה מהמוקס (ע"י העלאת הסיגנל DIN_{out}) ושומר אותו ב- R_x.

- עבור פקודת mv, העתקה מרגיסטר אחד לשני בזיכרון, (opcode 001) $R_x <= R_y$.

T_1 – המעבד בורר את R_y מהמוקס (ע"י העלאת הסיגנל $R0 \dots R7_{out}$) ושומר אותו לתוך R_x.

- עבור פקודת add , חיבור (opcode 010) או עבור פקודת sub, (opcode 011) בין ערכים השמורים בשני רגיסטרים צריך 4 מחזורי שעון. והפקודה נקבעת כך: $R_x = R_x \pm R_y$

T_1– המעבד בורר את R_x מהמוקס ושומר אותו ברגיסטר A שנכנס לALU.

T_2– המעבד בורר את R_y מהמוקס ומבצע את החיבור/חיסור ואת התוצאה הוא כותב לרגיסטר G שנכנס למוקס.

T_3– המעבד בורר את G מהמוקס ושומר אותו ב- R_x.

- עבור פקודת ספירת כמות האחדות במילה השמורה ברגיסטר והכנסתו לרגיסטר אחר (opcode 100) . צריך 3 מחזורי שעון. $R_y <= number\ of\ ones(R_x)$.

T_1– המעבד בורר את R_x מהמוקס ומכניס אותו אל תוך מודל העזר שסופר את כמות האחדות, ואת מספר האחדות שומרים ברגיסטר reg_ones.

T_2– המעבד בורר את reg_ones מהמוקס ושומר אותו ב- R_y.

בסוף כל פקודה אנחנו גם מעלים את הסיגנל Done.

לאחר שהסברנו את המבנה הכללי של המעבד נתחיל להסביר את המודלים השונים שיצרנו.

Processor

זהו המודל הראשי של המעבד (top) והוא מכיל את מימוש הרגיסטרים והחוטים של המערכת, קריאה לכל מודלי העזר וכן את מכונת המצבים של המעבד. המודל בנוי כך שבהתחלה אנחנו מגדירים את כל החוטים הפנימיים שנצרכים לנו, לאחר מכן את מכונת המצבים שהיא ליבו של המעבד ולבסוף קוראים למודלי העזר שלנו.

הכניסות

שעון/Clock - שעון המערכת. מחובר ל-KEY_1 בכרטיס ובכך אנחנו קובעים את תדר המעבד.

איתחול/Resetn - כפתור האיתחול. מחובר ל-KEY_0 בכרטיס.

הפקודה/Din- הדאטה שמכניסים למערכת והוא וקטור של 9 ביטים. מחובר ל- SW_{8-0} בכרטיס.

הרץ/Run – התחלת הפקודה. מחובר ל- SW_9 בכרטיס.

המוצאים

המוצא של המוקס/BusWires - המוצא שנבחר על ידי המוקס והוא וקטור של 9 ביטים. מחובר ל- $LEDR_{8-0}$ בכרטיס.

'סיימנו'/Done - מעלה 1 כאשר מסיימים פקודה. מחובר ל- $LEDR_9$ בכרטיס.

עכשיו נתחיל להסביר את כל קטעי הקוד.

בתחילת המודל אנחנו מגדירים את כל החוטים הפנימיים של המערכת והמצבים השונים. וכן מתרגמים את החלקים השונים של הדאטה לפקודה עצמה, R_x וR_y על ידי מודל שניתן לנו dec3to8 שהוא בעצם מפענח בינארי (Decoder).

מצורף הקוד :

```
module proc (DIN, Resetn, Clock, Run, Done, BusWires);
                input wire [8:0] DIN;
                input wire Resetn, Clock, Run;
                output reg Done;
                output wire [8:0] BusWires;


                parameter T0 = 2'b00, T1 = 2'b01, T2 = 2'b10, T3 = 2'b11;

                reg [1:0] Tstep_Q;//cs
                reg [1:0] Tstep_D;//ns
                wire [2:0] I;//instruction
                wire [7:0] Xreg;// en the reg that we write to
                wire [7:0] Yreg;

                //the register enablers
                reg [7:0] Rin;
                reg IRin, Ain, Gin, add_sub, O_in;

                //the mux selectors
                reg DIn_out, Gout, ones_out;
                reg [8:0] Rout;

                //the input to the mux
                wire [8:0] R0;
                wire [8:0] R1;
                wire [8:0] R2;
                wire [8:0] R3;
                wire [8:0] R4;
                wire [8:0] R5;
                wire [8:0] R6;
                wire [8:0] R7;
                wire [8:0] ones;
                wire [8:0] G;
                wire [8:0] num_of_1;//output of ones


                //internal wires for reg usage
                wire [8:0] A;//input to the addsub
                wire [8:0] IR;// the instruction that is being executed (the IR reg)
                wire [8:0] Alu_G;//wire between addsub and G


                assign I = IR[8:6];
                dec3to8 decX (IR[5:3], 1'b1, Xreg);
                dec3to8 decY (IR[2:0], 1'b1, Yreg);
```

לאחר מכן נדאג למעבר בין המצבים כלומר בין ה'מצב הנוכחי' (Tstep_Q) ל'מצב הבא' (Tstep_D), וזה בנוי בצורה הרגילה על פי שעון המערכת. כאשר אנחנו לוחצים על כפתור האתחול אנחנו מחזירים את ה'מצב הנוכחי' (Tstep_Q) לT0 ובכך מפסיקים את הריצה של הפקודה.

מצורף הקוד:

```
// Control FSM flip-flops
always @(posedge Clock, negedge Resetn)
   begin
   if (!Resetn)
      begin
         Tstep_Q <= T0;
      end
   else
      begin
         Tstep_Q <= Tstep_D;
      end
   end
```

הגדרנו ארבע מצבים שונים שהם ארבעת מחזורי השעון שנצרך לפקודה הארוכה ביותר. המצבים לא משתנים עד שנקבל את הפקודה 'להריץ' (Run) על ידי לחיצה על כפתור, לאחר הלחיצה המצב הנוכחי עובר ממחזור למחזור עד שמסיימים את הפקודה ומעלים את האות 'סיימנו' (Done) ובכך חוזרים למחזור הראשון T_0.

מצורף הקוד:

```
// Control FSM state table
always @(Tstep_Q, Run, Done)
   begin
   case (Tstep_Q)
      T0: // data is loaded into IR in this time step
         if (~Run)
            Tstep_D = T0;
         else
            begin
            Tstep_D = T1;
            end
      T1:
      begin

         if(Done)
            Tstep_D = T0;
         else
            Tstep_D = T2;
      end
      T2:
         if(Done)
            Tstep_D = T0;
         else
            Tstep_D = T3;
      T3:
         Tstep_D = T0;
   endcase
   end
```

במהלך כל מחזור אנחנו מעלים את אותות הבקרה של המערכת הנצרכים לאותו מחזור (לכתוב לתוך רגיסטר או לקרוא מתוך המוקס), לפי הפקודה שאנחנו מבצעים. כאשר אנחנו מכבים את כל האותות בקרה בתחילת הalways ומעלים אותם לפי הצורך.

מצורף הקוד:

```
// Control FSM outputs
   always @(Tstep_Q or I or Xreg or Yreg)
   begin
         Rin <= 8'b0;
         Ain <= 1'b0;
         Gin <= 1'b0;
         Rout <= 9'b0;
         DIn_out <= 1'b0;
         Gout <= 1'b0;
         add_sub <= 1'b0;
         Done <= 1'b0;
         O_in <= 1'b0;
         ones_out <= 1'b0;


   case (Tstep_Q)
         T0: begin  // store DIN in IR in time step 0 begin
               IRin <= 1'b1;//en for register IR
            end

            T1: //define signals in time step 1
               begin
               IRin <= 1'b0;//closes the instruction reg

               case (I)
                     3'b000:
                           begin
                           Rin <= Xreg;
                           Rout <= Yreg;
                           Done <= 1'b1;
                           end
                     3'b001:
                            begin
                            Rin <= Xreg;
                            DIn_out <= 1'b1;
                            Done <= 1'b1;
                            end
                     3'b010:
                            begin
                            Rout <= Xreg;
                            Ain <= 1'b1;
                            end
                     3'b011:
                            begin
                            Rout <= Xreg;
                            Ain <= 1'b1;
                            end
                     3'b100:
                            begin
                            Rout <= Xreg;
                            O_in <= 1'b1;
                            end
               endcase

               end
```

```
                        T2: //define signals in time step 2
                           case (I)
                                 3'b010:
                                        begin
                                        Rout <= Yreg;
                                        Gin <= 1'b1;
                                        end
                                 3'b011:
                                        begin
                                        Rout <= Yreg;
                                        Gin <= 1'b1;
                                        add_sub <= 1'b1;
                                        end
                                 3'b100:
                                        begin
                                        Rin <= Yreg;
                                        ones_out <= 1'b1;
                                        Done <= 1'b1;
                                        end
                           endcase

                        T3: //define signals in time step 3
                           case (I)
                                 3'b010:
                                        begin
                                        Gout <= 1'b1;
                                        Rin <= Xreg;
                                        Done <= 1'b1;
                                        end
                                 3'b011:
                                        begin
                                        Gout <= 1'b1;
                                        Rin <= Xreg;
                                        Done <= 1'b1;
                                        end
                           endcase

                     endcase
                  end
```

לבסוף הגדרנו את כל הרגיסטרים שהשתמשנו במהלך התוכנית וכן קראנו למודלי עזר שיצרנו ויתבארו בהמשך.

```
                regn reg_0 (BusWires, Rin[0], Clock, R0);
                regn reg_1 (BusWires, Rin[1], Clock, R1);
                regn reg_2 (BusWires, Rin[2], Clock, R2);
                regn reg_3 (BusWires, Rin[3], Clock, R3);
                regn reg_4 (BusWires, Rin[4], Clock, R4);
                regn reg_5 (BusWires, Rin[5], Clock, R5);
                regn reg_6 (BusWires, Rin[6], Clock, R6);
                regn reg_7 (BusWires, Rin[7], Clock, R7);

                regn reg_A (BusWires, Ain, Clock, A);
                regn reg_G (Alu_G, Gin, Clock, G);
                regn reg_IR (DIN, IRin, Clock, IR);
                regn reg_ones (num_of_1, O_in, Clock, ones);

                AddSub alu(.regA(A), .Bus(BusWires), .add_sub(add_sub), .regG(Alu_G));
                mux_eleven2one mux(DIN, R0, R1, R2, R3, R4, R5, R6, R7, G, ones, Rout,
                                   Gout, DIn_out, ones_out, BusWires);
                ones ones1(.Rx(BusWires), .num_of_1(num_of_1));


endmodule
```

לצורך בדיקת המעבד הוצאנו למוצא גם את הרגיסטר הראשון והשני כדי שנוכל לקרוא את המילה השמורה בהם ולוודא את ביצוע הפקודות. כמו כן נשים לב שכאשר אנחנו לוחצים על כפתור האתחול אנחנו בעצם מעבירים את מכונת המצבים למצב הראשון T_0, והפקודה נכנסת לרגיסטר ונשמרת שם וכאשר אנחנו מפסיקים ללחוץ על כפתור האתחול המעבד ממשיך למצב T_1 ובכך אנחנו 'מדלגים' על T_0 בפקודה הראשונה.

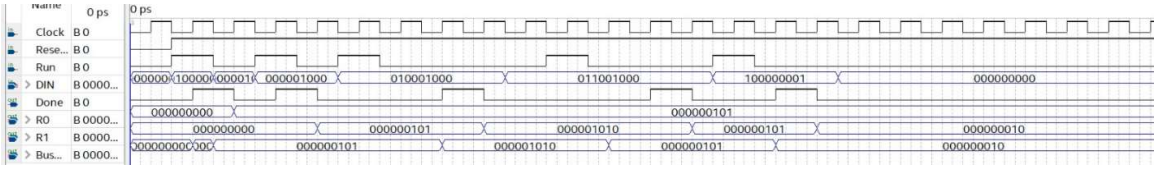
בסימולציה הכנסנו את המספר 5 לתוך רגיסטר 0 לאחר מכן העברנו את המספר מR_0 ל-R_1 (עכשיו המספר 5 נמצא בתוך שניהם לפי הגדרת הפקודה).

אחרי זה חיברנו בין שני הרגיסטרים כך: $R_1 = R_1 + R_0$ ונצפה שיהיה 10 בR_1 וR_0 יהיה ללא שינוי.

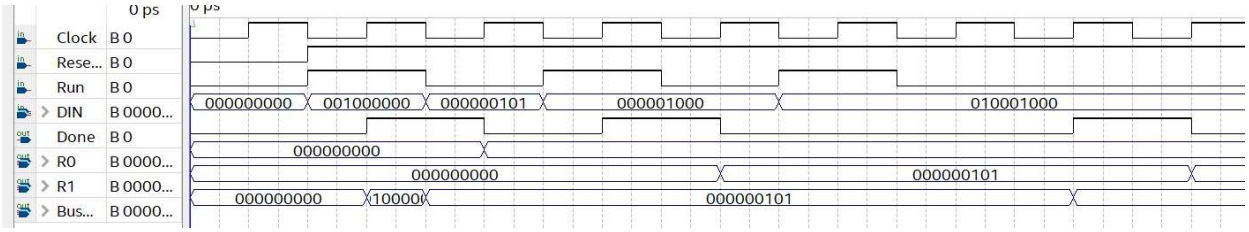
נחסר בין שניהם כך: $R_1 = R_1 - R_0$. נצפה שבשני הרגיסטרים יהיה שוב את המספר 5.

לבסוף נספור את כמות האחדים שנמצא בR_0 ונכניס את המספר הזה ל R_1. כלומר בR_1 יהיה את המספר 2.

וקיבלנו לפי מה שציפינו:

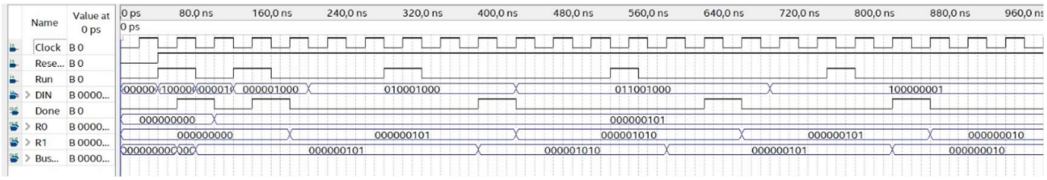


צילום תקריב של ההתחלה:

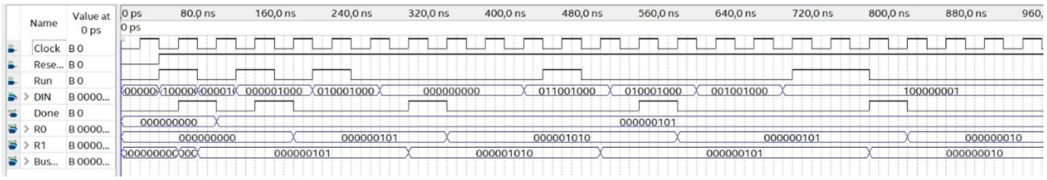


נבדוק עכשיו מספר מקרי קצה, כדי להקל על ההבנה האם המעבד הצליח או לא השתמשנו באותם ערכים ובסדר הפקודות כמו שיש למעלה.

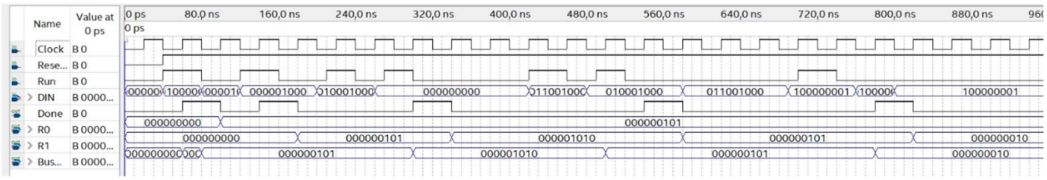
השארנו את פקודת ספירת האחדות וכן את פקודת החיבור בלי ללחוץ על 'הרץ' ורואים שכל עוד לא מורים למעבד 'הרץ' הוא לא מריץ את הפקודה.



הכנסנו פקודת אפסים בDin תוך כדי החיבור וכן הכנסו פקודת add ופקודת mvi תוך כדי החיסור.



בנוסף לסימולציה לעיל הכנסנו מספר 'הרץ' תוך כדי שהמעבד באמצע פקודה אחרת ואנחנו מצפים שלא ישתנה כלום.



mux_eleven2one

חלק בסיסי במעבד שלנו הוא המוקס שתפקידו לבחור מאיפה אנחנו לוקחים את המידע שלנו לפקודות השונות. תפעול המוקס יתבצע על ידי מכונת המצבים.

בחרנו לממש את המוקס על ידי כמה תנאים (IF) ולא על ידי וקטור ארוך שכולל את הכול (שהיה קל יותר לוודא נכונות באופן זה), כדי שהמודל עצמו יהיה קריא יותר והסלקטור ברור יותר וכן יהיה מובן יותר במכונת המצבים עצמה איזה סיגנל אנחנו מעלים או מורידים ולמה.

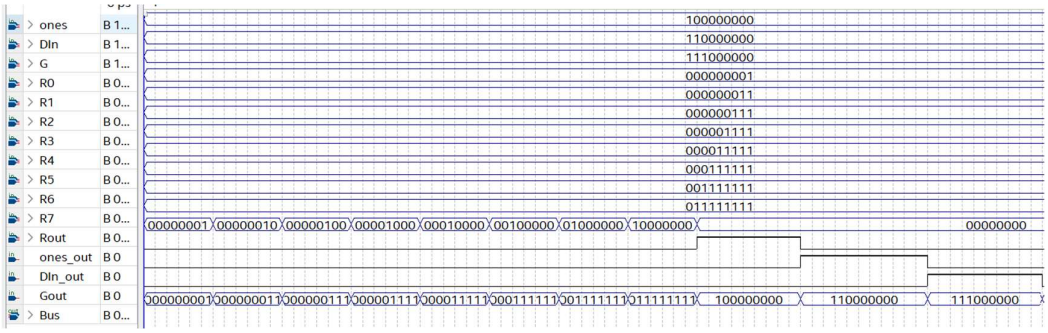
כניסות- מקבל את כל הכניסות שלו מהמודל הראשי proc. הכניסות מחולקות לשניים הכניסות והסלקטור.

מוצא/Bus- זהו המוצא המרכזי של המעבד, כמו כן משתמשים בו לצורך בחירת איזה מידע יכתב לרגיסטר וכן לבחירת מידע שייכנס לALU או לספירת האחדות.

מצורף הקוד שכתבנו :

```
module mux_eleven2one(
                    // inputs to the mux
                     input wire [8:0] DIn,
                     input wire [8:0]R0,
                     input wire [8:0]R1,
                     input wire [8:0]R2,
                     input wire [8:0]R3,
                     input wire [8:0]R4,
                     input wire [8:0]R5,
                     input wire [8:0]R6,
                     input wire [8:0]R7,
                     input wire [8:0]G,
                     input wire [8:0]ones,
                     //selectors
                     input wire [7:0]Rout,
                     input wire Gout,
                     input wire DIn_out,
                     input wire ones_out,

                     output reg [8:0] Bus
                     );

                    always  @(Rout, Gout, DIn_out)
                    begin
                       if(Gout)
                          Bus = G;

                       else if (DIn_out)
                          Bus = DIn;

                       else if (ones_out)
                             Bus = ones;
                       else
                          begin
                          case(Rout)
                             8'b00000001: Bus = R0;
                             8'b00000010: Bus = R1;
                             8'b00000100: Bus = R2;
                             8'b00001000: Bus = R3;
                             8'b00010000: Bus = R4;
                             8'b00100000: Bus = R5;
                             8'b01000000: Bus = R6;
                             8'b10000000: Bus = R7;
                             endcase
                          end
                    end

endmodule
```

לצורך בדיקת המודל הכנסנו לסימולטור ובדקנו את כל הכניסות השונות. מקרי קצה כגון ששני סיגנלים יהיו למעלה, אכן יכולים להוות בעיה למערכת אבל אנחנו דואגים במודל הראשי להכניס קלט תקין בלבד ולכן מקרים כגון אלו לא אמורים להתקיים.



AddSub

זה הALU של המעבד שלנו, המודל שאחראי על הפקודות חיבור וחיסור. אנחנו מקבלים שני רגיסטרים וסיגנל המורה האם לבצע חיבור או חיסור, והיחידה מבצעת כך את הפקודה:
$regG = regA \pm Bus$.

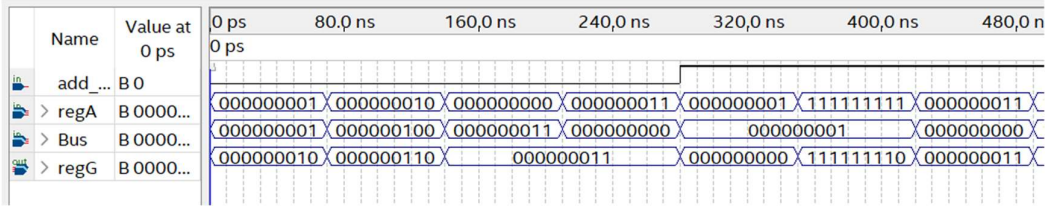
כניסות- מקבל כניסה אחת מהרגיסטר A וכניסה אחת מהמוקס.

מוצא/regG - יכנס לתוך המוקס וייכתב אל תוך הרגיסטר R_x זמן מחזור מאוחר יותר.

מצורף הקוד שכתבנו:

```
1  module AddSub(
2                 input wire [8:0] regA,
3                 input wire [8:0] Bus,
4                 input wire add_sub, //0 for add, 1 for sub
5
6                 output reg [8:0] regG
7                 );
8
9                  always @(regA)
10                    begin
11                    if (~add_sub)
12                       regG <= regA + Bus;
13                    else
14                       regG <= regA - Bus;
15                    end
16 endmodule
17
18
```

בדקנו את תקינות המודל והכנסנו לסימולטור מקרי קצה מסוימים כגון חיבור וחיסור של 0 וכן כאשר רק אחד מהרגיסטרים משתנה וקיבלנו תוצאות כמצופה. יש להעיר שבהתאם לדרישות לא התייחסנו למקרה של גלישה או קבלת מספר שלילי.

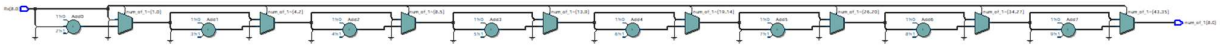


ones

המודל מקבל רגיסטר ומחזיר את כמות האחדות שיש בו.

מימשנו את זה על ידי לולאה פשוטה שעוברת על כל הביטים ברגיסטר וסוכמת את כמות האחדות. מכיוון שמדובר בשפת תיאור חומרה (HDL) אין באמת דבר כזה לולאה המבצעת איטרציה אחרי איטרציה, שהרי מדובר בחומרה פיזית. אבל עדיין בורילוג אפשר לכתוב לולאות, וורילוג בעצם מתרגם את הלולאה כאל דרך מקוצרת לתאר שכפול של חומרה בהתאם לכמות האיטרציות הדרושות.

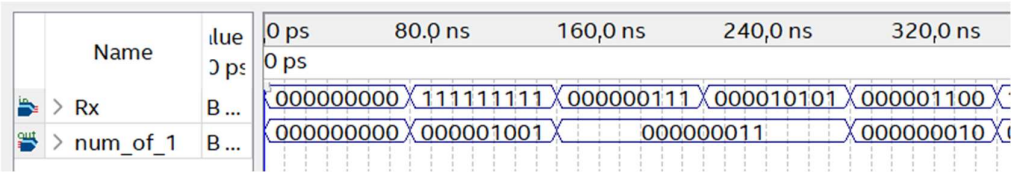
מצורף התרשים בלוקים של הקוד כדי להסביר בצורה וויזואלית את דרך פעולת הלולאה:



מצורף הקוד שכתבנו:

```
module ones(input wire [8:0] Rx,
            output reg [8:0] num_of_1);
            integer i;
            always @(Rx)
               begin
                  num_of_1 = 8'b0;

                     for(i=0; i<9; i=i+1)
                       if(Rx[i]==1'b1)
                          num_of_1 = num_of_1 + 8'b1;

                end
endmodule
```

מצורפת סימולציה לבדיקת התקינות של המודל, בדקנו מקרי קצה מסוימים ורואים שהלולאה תקינה ומקבלים תוצאות כמצופה.



מצורף קישור לסרטון שבו אנו מציגים את ביצועי המעבד על הכרטיס:

https://drive.google.com/file/d/1AvsmD6cIrcExSCZz4aY6ZorErwlYfbMz/view?usp=sharing