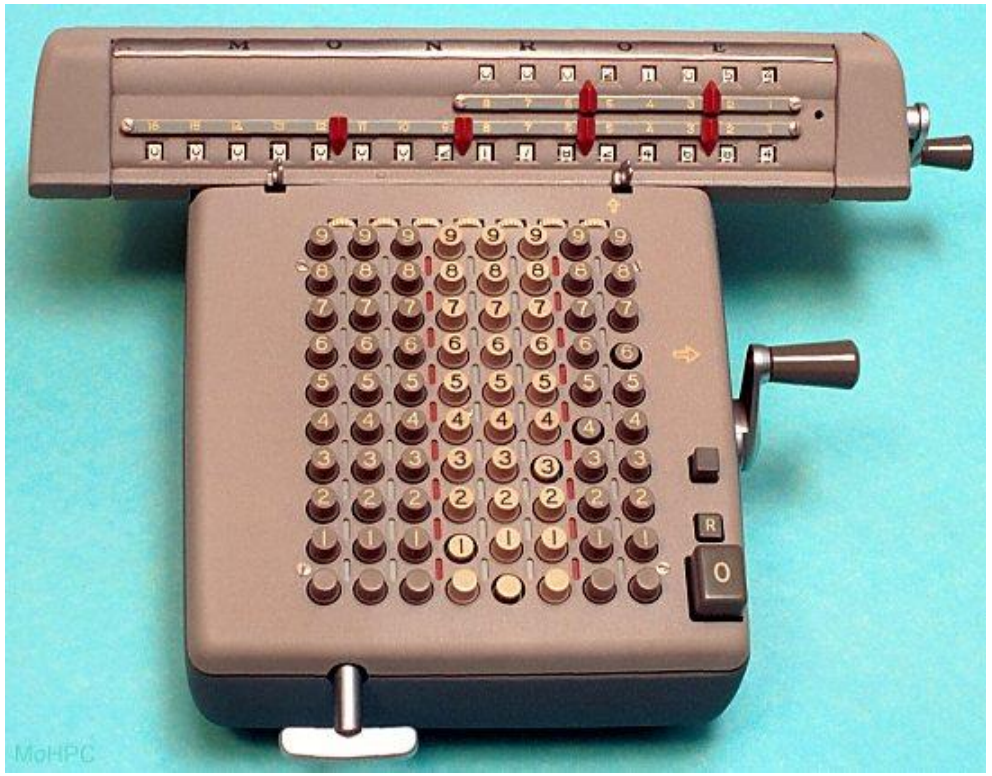
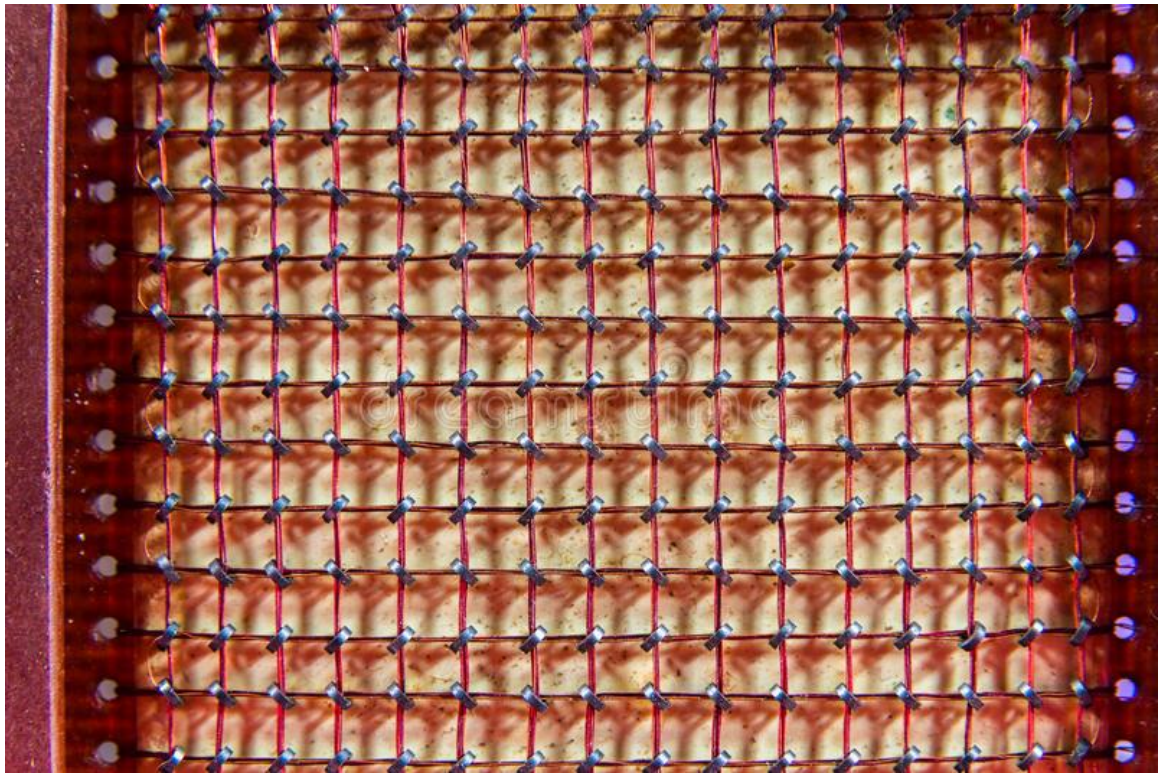
EE 044252: Digital Systems and Computer Structure

Spring 2018

Lecture 4: *Arithmetic and Memory*





EE 044252: Digital Systems and Computer Structure

Topic	wk	Lectures	Tutorials	Workshop	Simulation
Arch	1	Intro. RISC-V architecture	Numbers. Codes		
Comb	2	Switching algebra & functions	Assembly programming		
	3	Combinational logic	Logic minimization	Combinational	
	4	Arithmetic. Memory	Gates		Combinational
Seq	5	Finite state machines	Logic		
	6	Sync FSM	Flip flops, FSM timing	Sequential	Sequential
	7	FSM equiv, scan, pipeline	FSM synthesis		
	8	Serial comm, memory instructions	Serial comm, pipeline		
μArch	9	Function call, single cycle RISC-V	Function call		
	10	Multi-cycle RISC-V	Single cycle RISC-V		Multi-cycle
	11	Interrupts, pipeline RISC-V	Multi-cycle RISC-V		
	12	Dependencies in pipeline RISC-V	Microcode, interrupts		
	13		Depend. in pipeline RISC-V		

Agenda

- Arithmetic logic circuits
  - Full adder
  - Ripple carry adder
  - Carry lookahead adder
  - RISC-V ALU
- Fault detection in combinational logic
- Memories
  - Latch, flip-flop
  - Clock
  - Metastability
  - Register
  - Shift register
  - RISC-V register file
  - Memory types

חיבור מספרים בינריים

carry

		1	1							
	1	0	1	1				11		-5
+	0	0	1	1	+	+	3		+	3
	1	1	1	0				14		-2

unsigned

Signed 2's

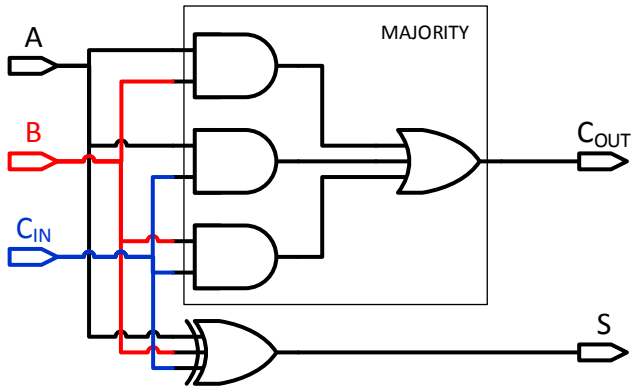
- בכל עמודה מחברים שלוש סיביות: נשא "נכנס" ושתים רשומות
- בכל עמודה מחשבים שתי תוצאות:

סיבית סכום העמודה וסיבית נשא "יוצא"

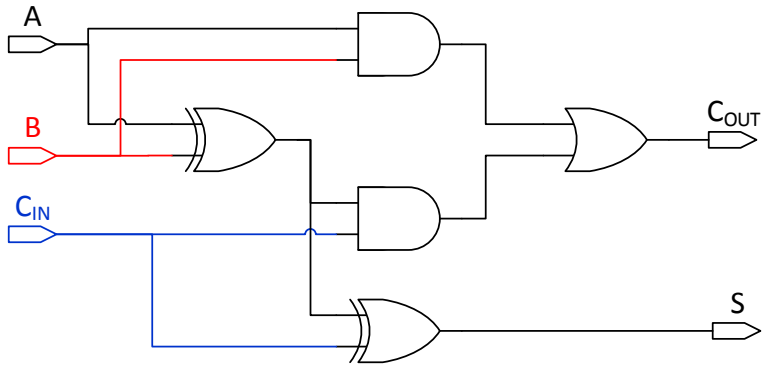
Full Adder (adds one column)

- [C_{OUT}, S] is the 2-bit count of the number of 1's: 0,1,2,3
- What is t_{PD}? What is $t_{PD}(C_{IN} \rightarrow C_{OUT})$?

A	B	C_{IN}	C_{OUT}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



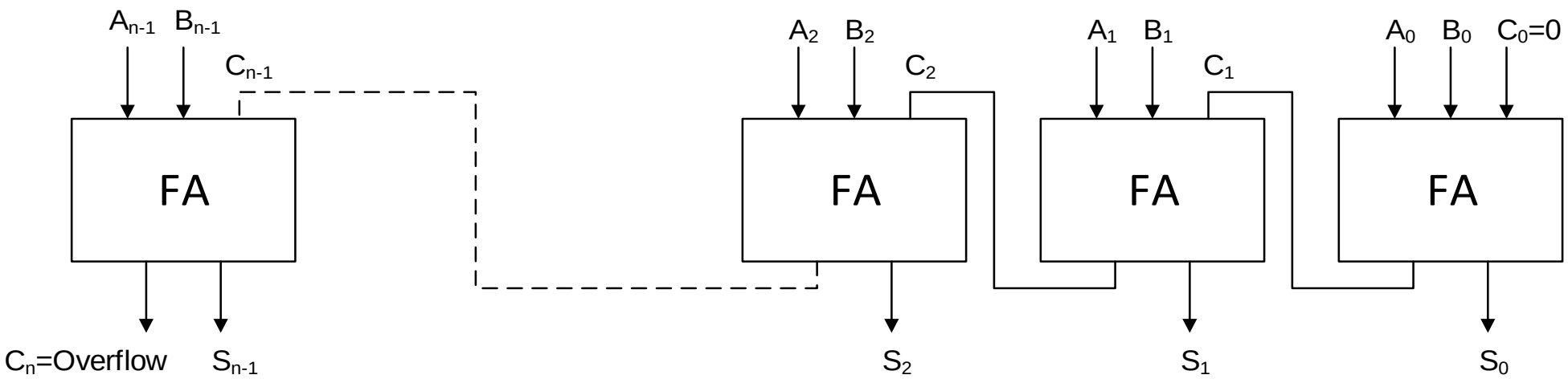
$$C = AB + AC + BC = AB + (A + B)C = AB + (AB + A \oplus B)C = AB + (A \oplus B)C$$



FA in Verilog

```
// full adder - logical
module FullAdder2(
   input a, b, cin,
   output wire cout, s // carry and sum
);
   assign s = a ^ b ^ cin ;
   assign cout = (a & b)|(a & cin)|(b & cin) ;
endmodule
```

Ripple Carry Adder (RCA)



- What is t_{PD}?
- Unit delay model? 3n
- All but FA_0 are 2n
- $t_{PD}=t_{PD}(C_{IN}\rightarrow C_{OUT})=2(n-1)+3\approx 2n$

RCA in Verilog: Behavioral model

```
// multi-bit RCA adder - behavioral
module RCA #(parameter n = 8) (
   input [n-1:0] a, b,
   input cin,
   output wire [n-1:0] s,
   output wire cout
);
   assign {cout, s} = a + b + cin ;
endmodule
```

- ***Verilog compiler can turn the code above into different gate-level implementations according to additional constraints of area, latency and power***

RCA in Verilog: bit-slices

```
// multi-bit adder - with vectors
module Adder2 #(parameter n = 8) (
 input [n-1:0] a, b,
 input cin,
 output wire [n-1:0] s,
 output wire cout
);

  wire [n:0]    c = {(a&b) | ((a^b) & c[n-1:0]), cin} ;
  assign [n-1:0] s = ((a^b) ^ c[n-1:0] ;
  assign cout = c[n] ;

endmodule
```

Carry Look-Ahead (CLA) Adder

$C = AB + (A \oplus B)C$

- ראינו כי A(i), B(i) מגיעים ברגע אפס, זמן רב לפני שיגיע C(i) (עבור i גדול)
- אם g(i)≡A(i)B(i)=1 אזי C(i+1)=1 ואין צורך לחכות ל-C(i)
  [g(i) קרויה Carry Generate]
  - לא באמת "מחכים" אבל כדאי להתחשב בזה בעת חישוב "אופורטוניסטי" של השהיה
- גם אם הנשא לא נוצר ביחידה i, הוא "יעבור" אם C(i)=1 וגם p(i)=A(i)+B(i)=1
  [p(i) קרויה Carry Propagate]
  - לצורך צמצום לוגי, כדאי לציין כי $p(i)_{g(i)=0}$=A(i)⊕B(i)
- התוצאה ביחידה i: C(i+1)=g(i)+p(i)C(i)
- אילו היינו מסתפקים בזאת, לא נוכל לצמצם את t_{PD} --כאשר ∀i:g(i)=0 זמן החישוב זהה לזה של RCA...

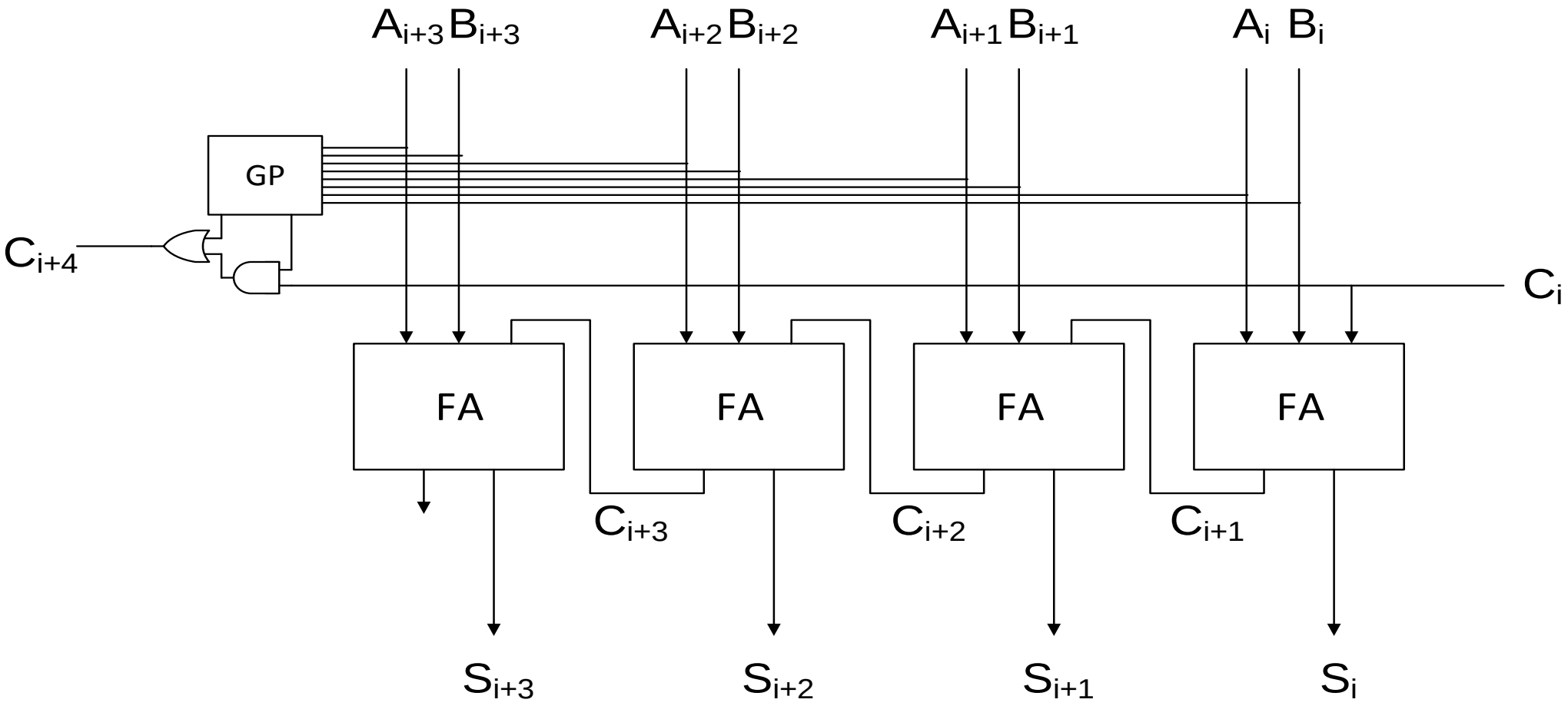
Carry Look-Ahead (CLA) Adder

- נתקדם עוד צעדים:

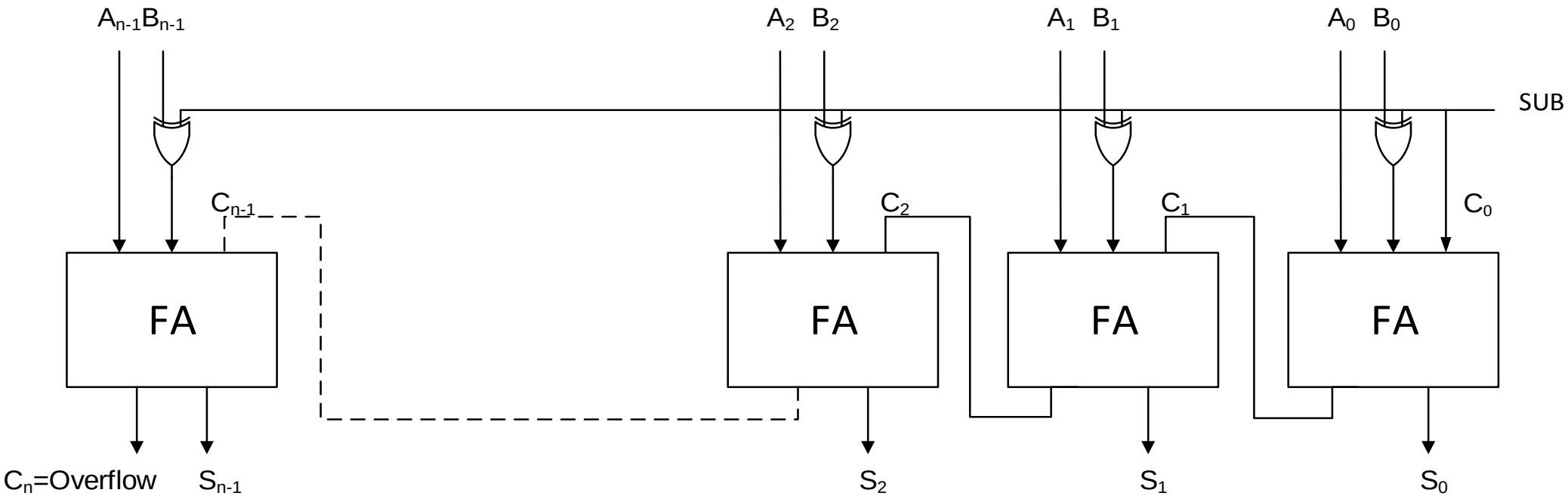
$$C_{i+1} = g_i + p_i C_i$$
$$C_{i+2} = g_{i+1} + p_{i+1} C_{i+1} = g_{i+1} + p_{i+1} g_i + p_{i+1} p_i C_i$$
$$C_{i+3} = \cdots$$
$$C_{i+4}$$
$$= g_{i+3} + p_{i+3} g_{i+2} + p_{i+3} p_{i+2} g_{i+1} + p_{i+3} p_{i+2} p_{i+1} g_i + p_{i+3} p_{i+2} p_{i+1} p_i C_i$$
$$= G_{i+4} + P_{i+4} C_i$$

- G, P תלויים רק בכניסות B,A בעלות אותם אינדקסים
- כעת C מסוים ממתין רק ל-C הנוצר ארבעה שלבים קודם לכן, וזמן ההשהיה הכולל הואץ פי ארבע

4-bit Carry Look-Ahead (CLA) Adder



Adder/Subtractor



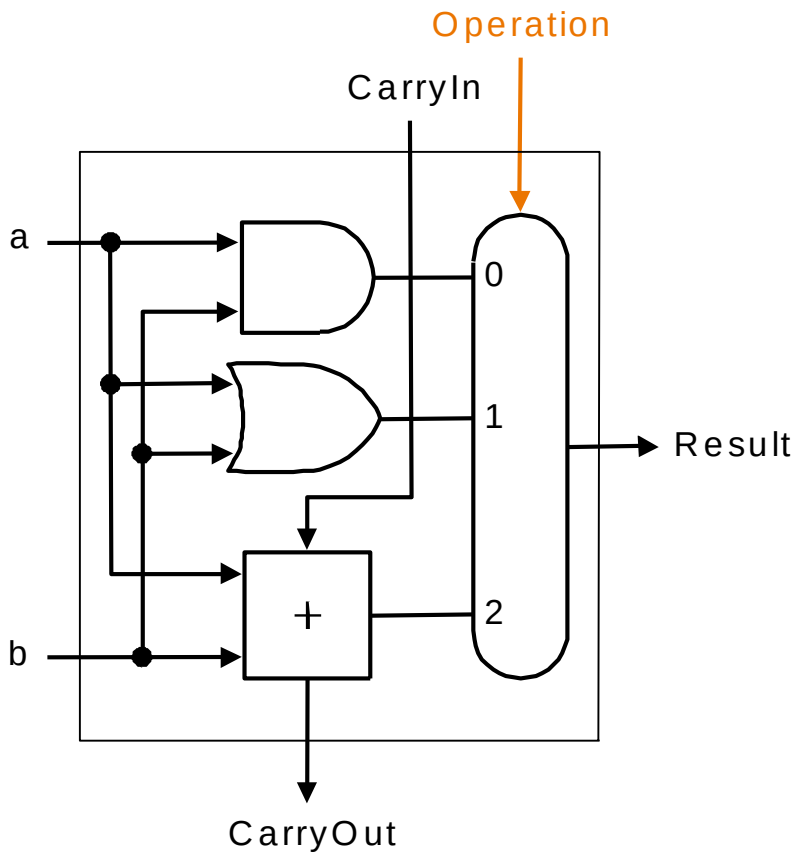
- Why does it work?

Agenda

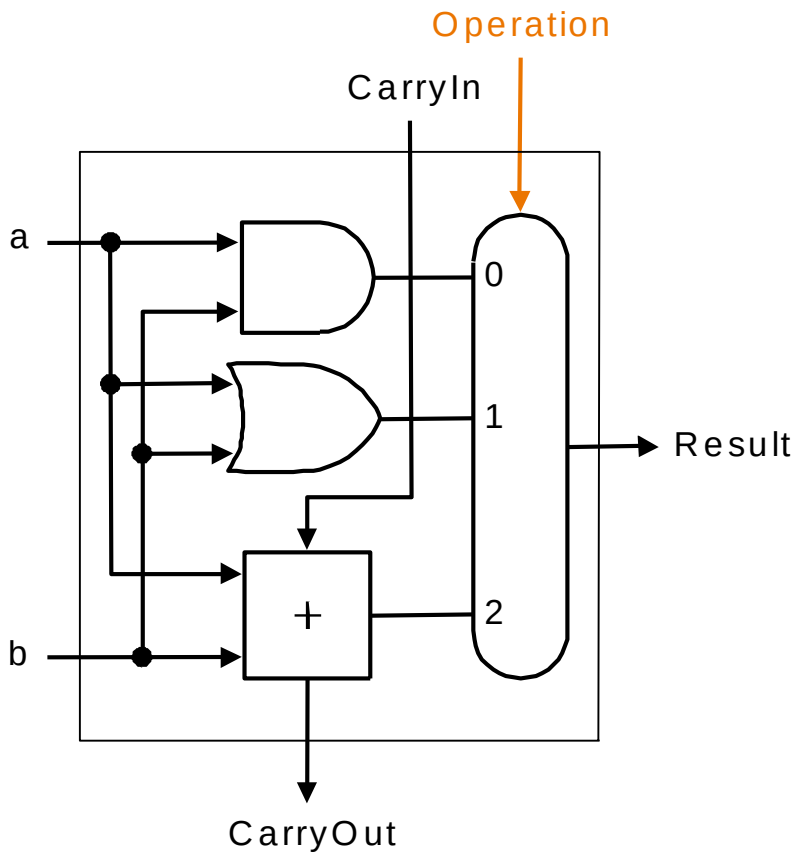
- Arithmetic logic circuits
  - Full adder
  - Ripple carry adder
  - Carry lookahead adder
  - **RISC-V ALU**
- Fault detection in combinational logic
- Memories
  - Latch, flip-flop
  - Clock
  - Metastability
  - Register
  - RISC-V register file
  - Shift register
  - Memory types

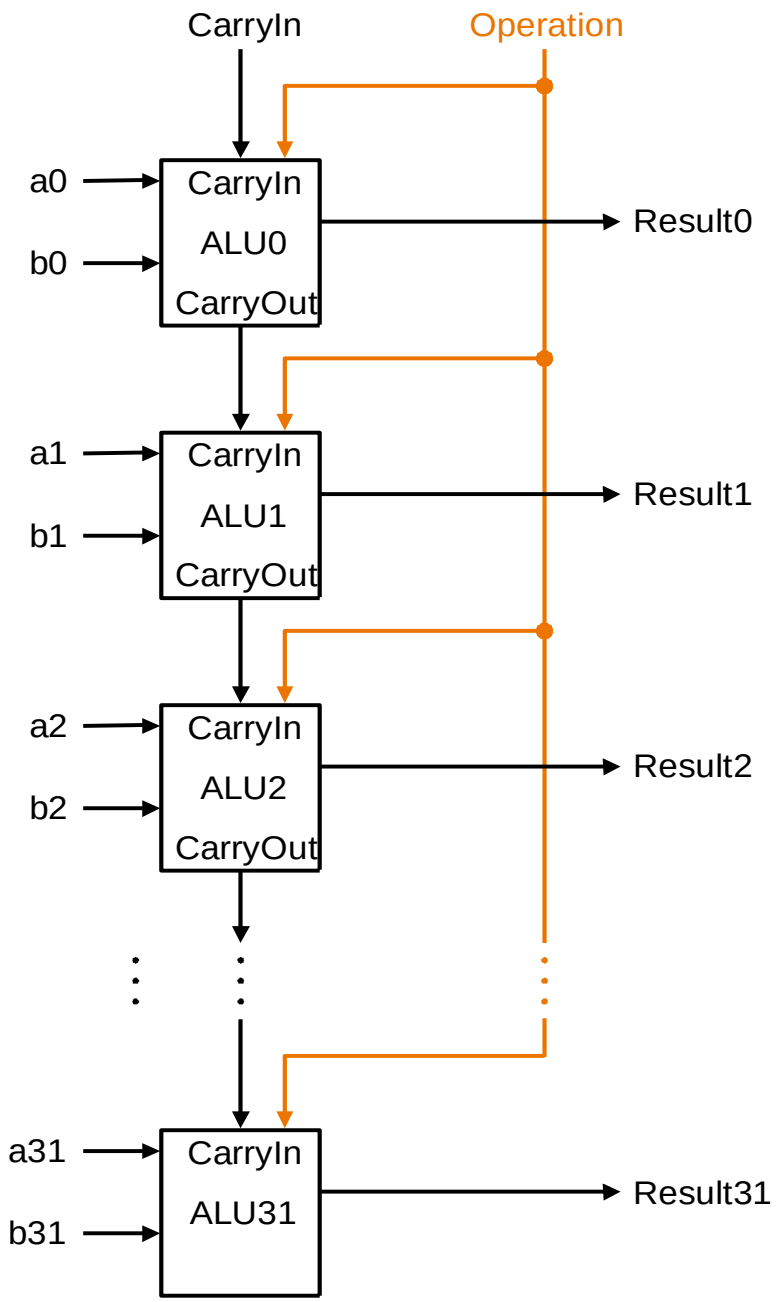
Arithmetic Logic Unit (ALU)

- נרחיב את המחבר ל-ALU, למשל בעל הפעולות הבאות:
  - A+B
  - A—B
  - And(A,B)
  - Or(A,B)
- מעגל עקרוני עבור סיבית (יציאה) אחת:

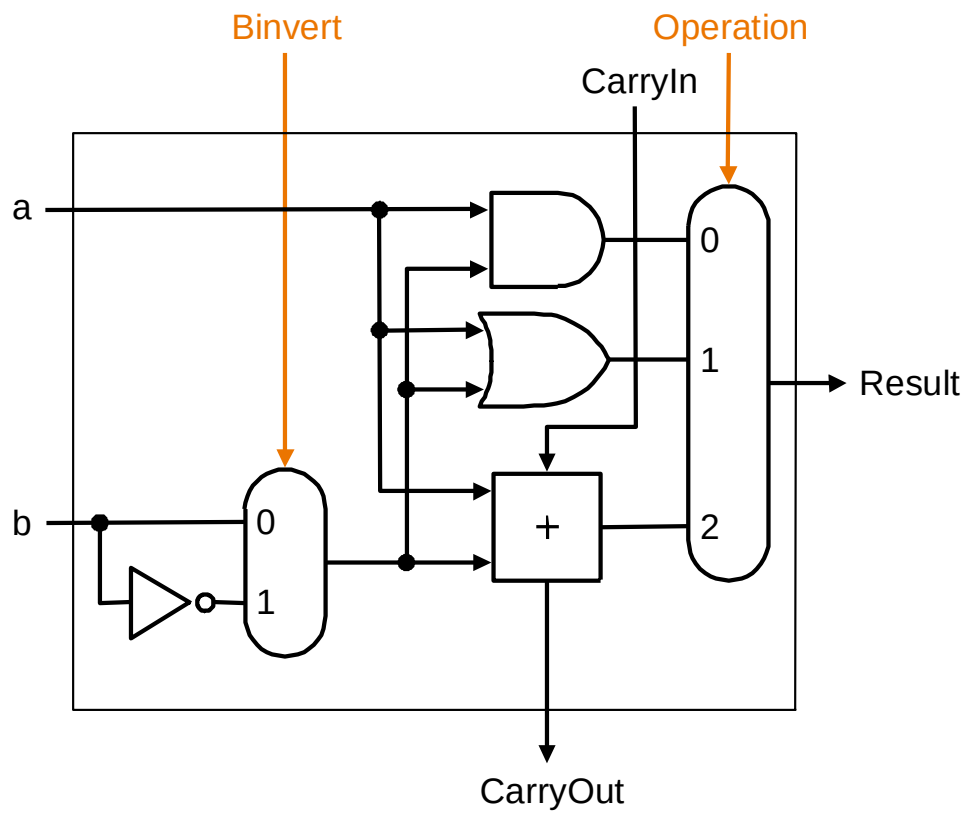


32-bit ALU


Operation
CarryIn
a
b
0
1
2
+
Result
CarryOut


CarryIn
Operation
a0
b0
CarryIn
ALU0
CarryOut
Result0
a1
b1
CarryIn
ALU1
CarryOut
Result1
a2
b2
CarryIn
ALU2
CarryOut
Result2
a31
b31
CarryIn
ALU31
Result31

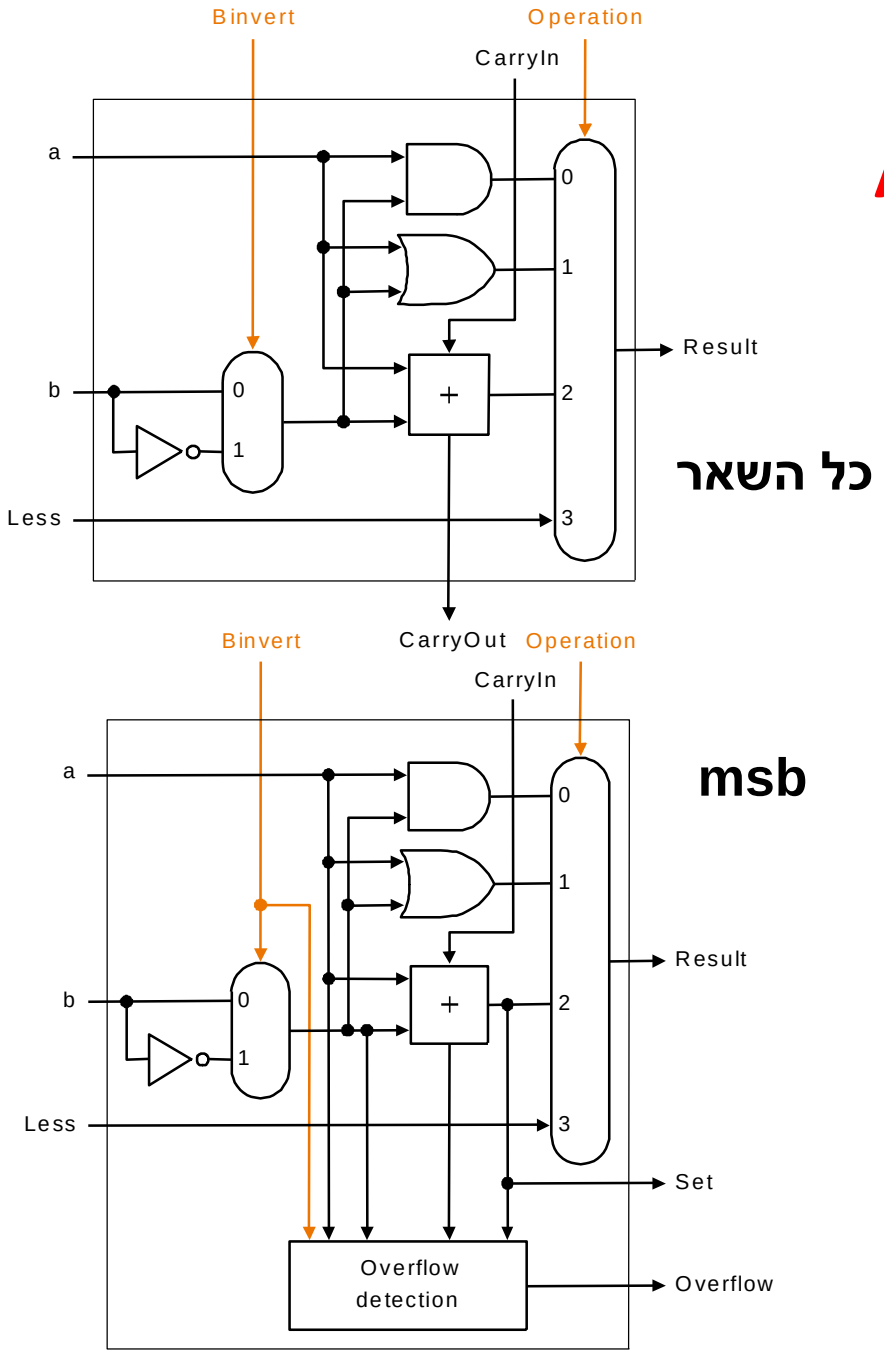
Add/Subtract ALU



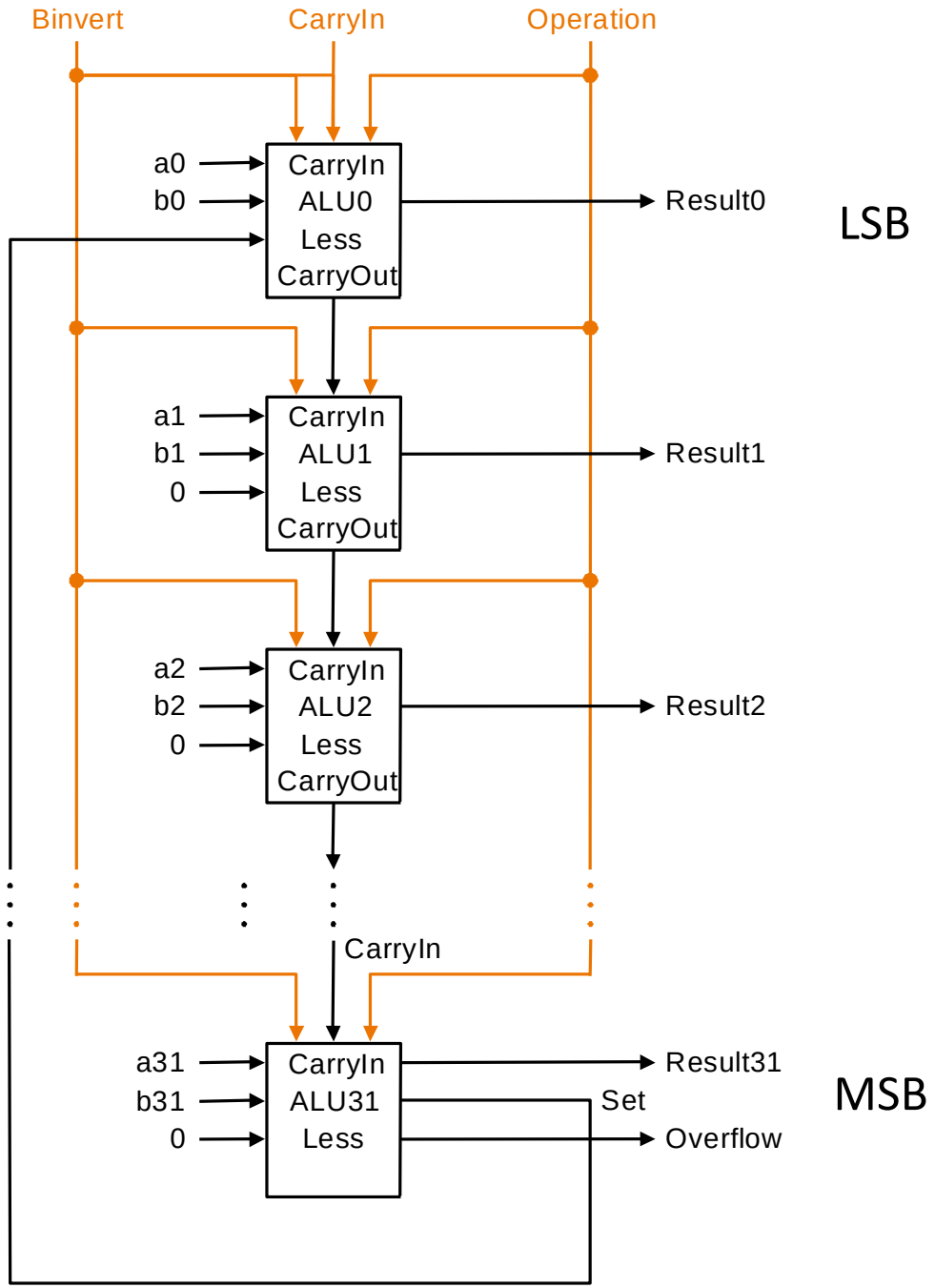
- למען הסימטריה, במחשב RISC-V אפשר גם להפוך את a

ALU with SLT (SET IF LESS THEN)

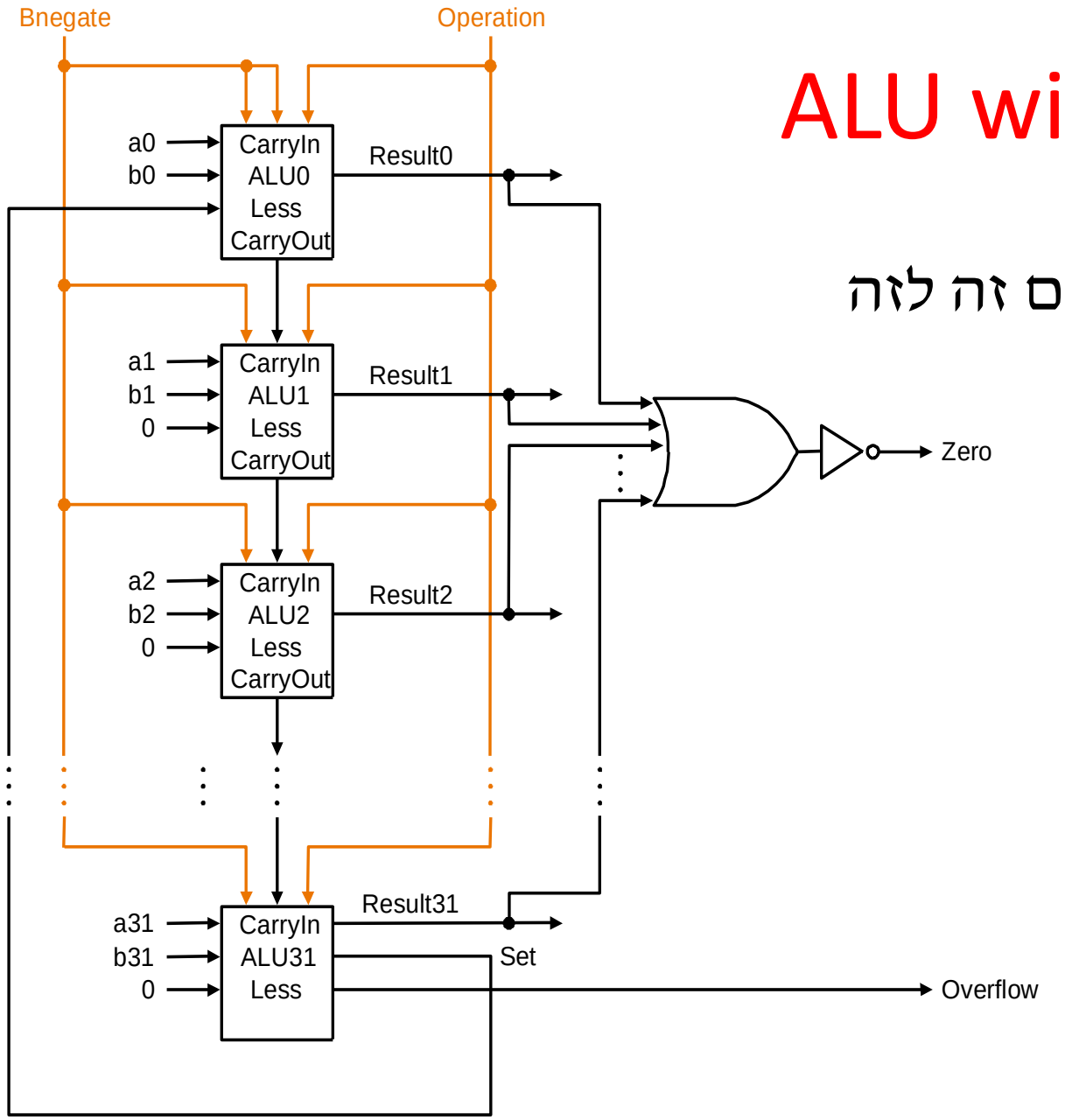
- נשתמש בחיסור לצורך השוואה מי גדול יותר
- פעולת SLT מציבה '0' בכל הסיביות חוץ מ-lsb
- ב-lsb מוצב 0/1 תלוי בסיבית msb של החיסור A-B
- ב-msb נוסיף חומרה לזיהוי גלישה



ALU with SLT

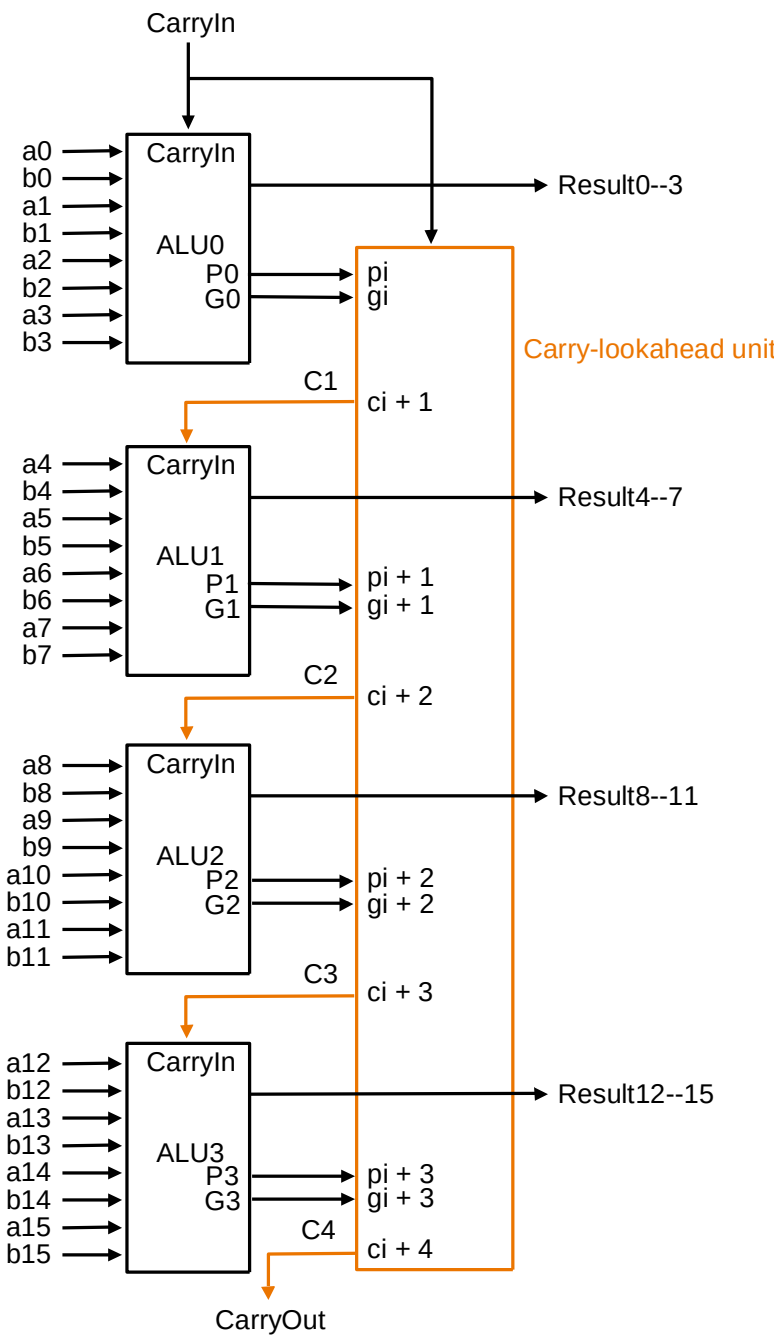

Binvert
CarryIn
Operation
a0
b0
CarryIn
ALU0
Less
CarryOut
Result0
LSB
a1
b1
0
CarryIn
ALU1
Less
CarryOut
Result1
a2
b2
0
CarryIn
ALU2
Less
CarryOut
Result2
CarryIn
a31
b31
0
CarryIn
ALU31
Less
Result31
Set
Overflow
MSB

ALU with SLT and equal-check



- אם החיסור אפס, האופרנדים היו שווים זה לזה

Adding CLA and other goodies



- CLA
- Multiplier
- Divider
- ועוד

Verilog ALU for RISC-V: Behavioral model

```
module RISCVALU (ALUctl,A,B,ALUOut,Zero);
   input [3:0] ALUctl ;
   input [31:0] A ,B;
   output reg [31:0] ALUOut ;        //what is reg?
   output Zero;

   assign Zero = (ALUOut==0 );       //Zero is true if ALUOut is 0
   always@(ALUctl, A, B) begin       //reevaluate if these change
      case (ALUctl )
         0:  ALUOut <= A & B ;
         1:  ALUOut <= A | B ;
         2:  ALUOut <= A + B ;
         6:  ALUOut <= A - B ;
         7:  ALUOut <=(A < B)? 1 : 0;
         12: ALUOut <=  ~(A | B);    // NOR
         default: ALUOut <= 0;
      endcase
   end
endmodule
```

Agenda

- Arithmetic logic circuits
  - Full adder
  - Ripple carry adder
  - Carry lookahead adder
  - RISC-V ALU
- **Fault detection in combinational logic**
- Memories
  - Latch, flip-flop
  - Clock
  - Metastability
  - Register
  - RISC-V register file
  - Shift register
  - Memory types

גילוי תקלות צירופיות

- תהליך הייצור של מעגלים ספרתיים איננו אידאלי : מדי פעם נופלות תקלות במעגלים

- Yield: Y= #Good Chips / #Fabricated Chips

$$\text{Chip Cost} = \frac{\text{Cost of fabrication + testing}}{\text{Y x \#Fabricated Chips}}$$

- Yield משתפר ככל שהטכנולוגיה החדשה מבשילה
- איך מוצאים אילו רכיבים תקולים? על ידי בדיקות (testing)
- מחיר הבדיקות תלוי בין היתר **במשך זמן הבדיקה** עבור רכיב יחיד
  - מכונת בדיקה מהירה עולה כמיליון דולר (תרגום : 50 סנט עלות כל דקה)

גילוי תקלות – מטרה

- אנו מחפשים דרכים למצוא את התקלות בכדי לסייע ליצרן להימנע מלמכור מוצרים פגומים
- במקרה הפשוט של מעגל צירופי, דרך אחת היא לעבור על פני כל טבלת האמת:
  - לייצר את כל 2^n הצירופים האפשריים (למעגל בעל n כניסות)
  - להשוות כל יציאה לערך הצפוי בטבלת האמת
  - בכל מקרה של שוני ביניהם, יש תקלה במעגל
- בדיקה כזו בדרך כלל ארוכה מדי
  - יש דרכים קצרות יותר להשגת אותה מטרה

סוגי בדיקות

- נגדיר שני סוגי בדיקת תקלות:
  - בדיקה ל**גילוי תקלות**, שמטרתה לקבוע האם המעגל תקין או לא
  - בדיקה ל**איתור התקלות**, שמטרתה לקבוע היכן בדיוק בתוך המעגל נמצאת התקלה
- לעיתים קשה לקבוע בדיוק מוחלט מה **מקום התקלה**, אך מספיק לקבוע באיזה חלק של המעגל קרתה התקלה על מנת שנוכל להחליף את כל החלק המקולקל
  - ונדע לתקן את המעגל המתאים בתוך השבב, בעת הייצור מחדש בעתיד

סוגי תקלות

- נגביל את הדיון רק לשני סוגים של תקלות:

1. "צומת תקועה ב-0" (s-a-0 , stuck-at-0) - תקלה לפיה חוט מסוים במעגל תקוע במצב לוגי 0 (או שהוא לוגית נראה כך)
2. "צומת תקועה ב-1" (s-a-1 , stuck-at-1) - תקלה לפיה חוט מסוים במעגל תקוע במצב לוגי 1 (או שהוא לוגית נראה כך)

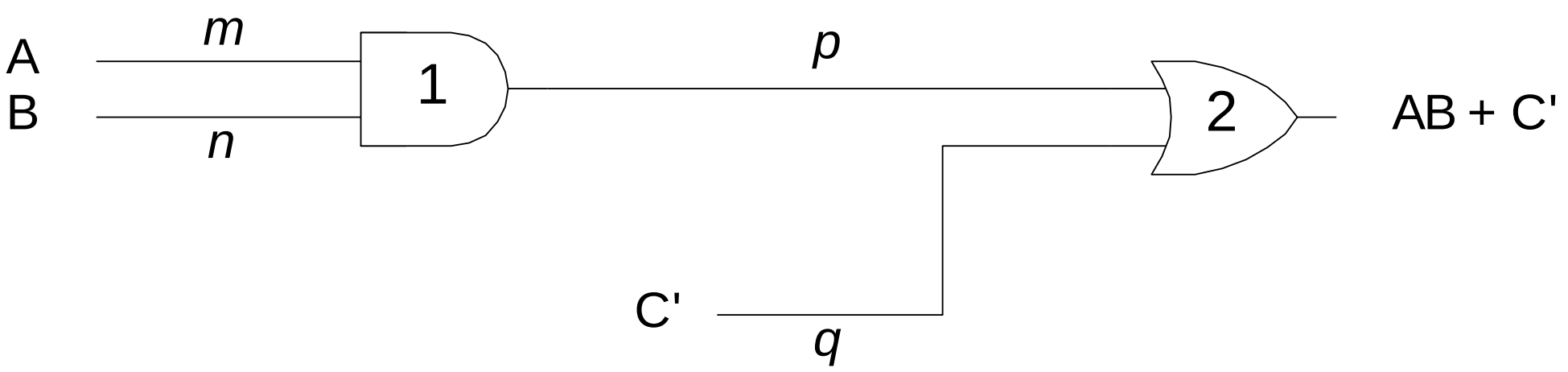
- הגבלה זו מוצדקת כי מרבית התקלות המעשיות אכן ניתנות לתיאור כאחת משתי התקלות הללו
- קיימים גם סוגים אחרים של תקלות:

i. "צומת מנותקת" (stuck open) – חוט מנותק
ii. "גישור" (bridging) – שני חוטים שונים מחוברים זה אל זה ואחד מהם קובע את הערך הלוגי של שניהם
iii. "השהייה" – שער מגיב הרבה יותר לאט מהצפוי

ניסוי ובדיקה לגילוי תקלות

- נבדיל בין
  "ניסוי לגילוי תקלות" ו"בדיקה לגילוי תקלה"
  - הניסוי מורכב ממספר בדיקות
  - בכל בדיקה מגישים למעגל צירוף אחד בלבד של הכניסות ובודקים את היציאות
  - מטרת הניסוי לבחון האם המעגל עובד נכון עבור כל קבוצת הבדיקות המוכלות בניסוי

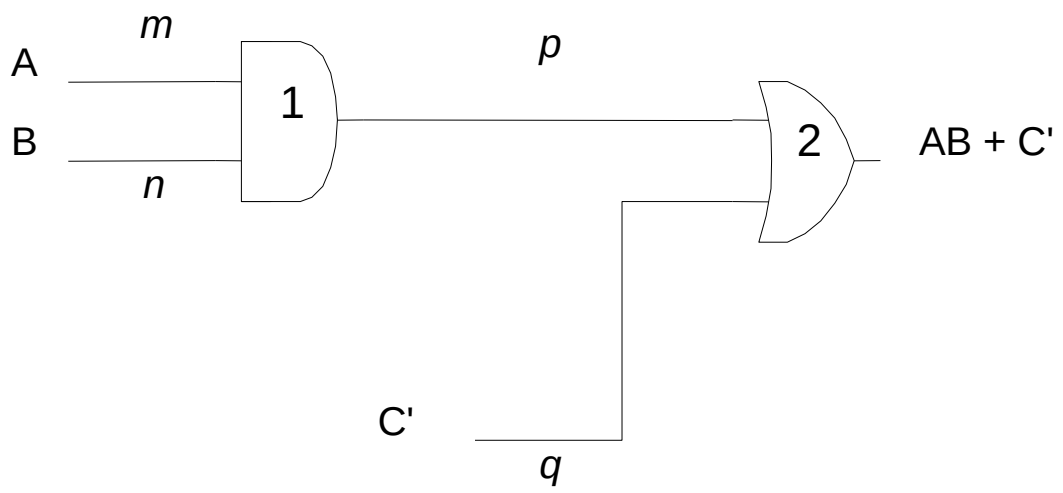
דוגמה



- לפונקציה שלוש כניסות, ולכן צריך 8 בדיקות לכל היותר
- נסמן, למשל, ב- *m*0 את התקלה שבה החוט *m* תקוע ב-0, וכן הלאה, ונקבל את טבלת התקלות להלן :

טבלת התקלות

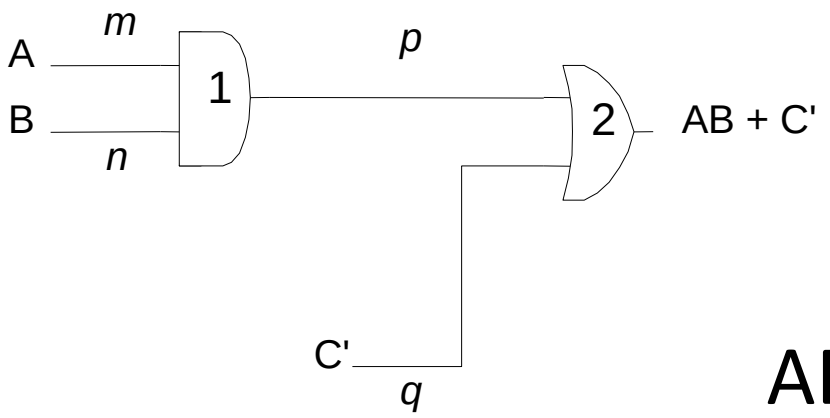
Inputs ABC	Possible Faults							
	m0	n0	p0	q0	m1	n1	p1	q1
000				x				
001							x	x
010				x				
011					x		x	x
100				x				
101						x	x	x
110								
111	x	x	x					



* single error is assumed

טבלת תקלות

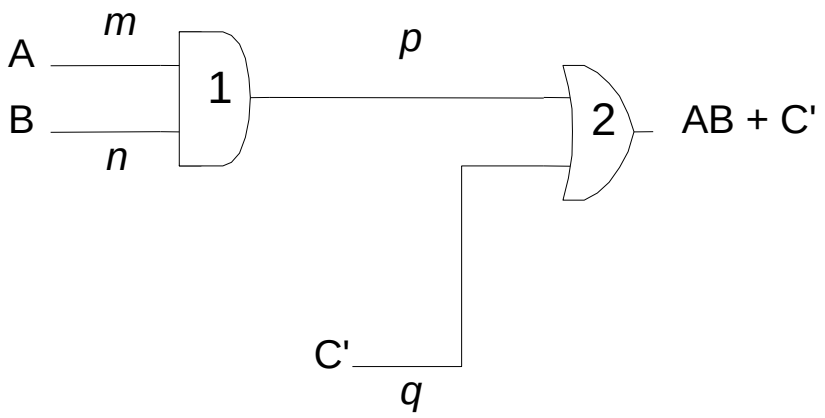
- נבנה טבלת תקלות ובה שורה לכל בדיקה אפשרית ועמודה לכל תקלה אפשרית
  - כאשר בדיקה מסויימת i מגלה תקלה מסויימת j , נכניס x במשבצת המתאימה (שורה i , עמודה j)
  - נמצא קבוצה מינימלית של שורות בטבלה כך שבכל עמודה יהיה x אחד לפחות
  - נאמר שקבוצה זו "מכסה" את טבלת התקלות



בניית טבלת התקלות

- נבחן למשל את צירוף הכניסות הראשון, ABC=000
  - היציאה אמורה להיות 1, קו *p* אמור להיות 0 וקו *q* אמור להיות 1
- אילו תקלות יכולות לגרום ליציאה להיות 0 במקום 1?
  - רק *s-a-0* *q*, ולכן נסמן *x* בעמודה המתאימה. באופן דומה משלימים את כל הטבלה

Inputs	Possible Faults							
ABC	m0	n0	p0	q0	m1	n1	p1	q1
000				x				
001							x	x
010				x				
011					x		x	x
100				x				
101						x	x	x
110								
111	x	x	x					

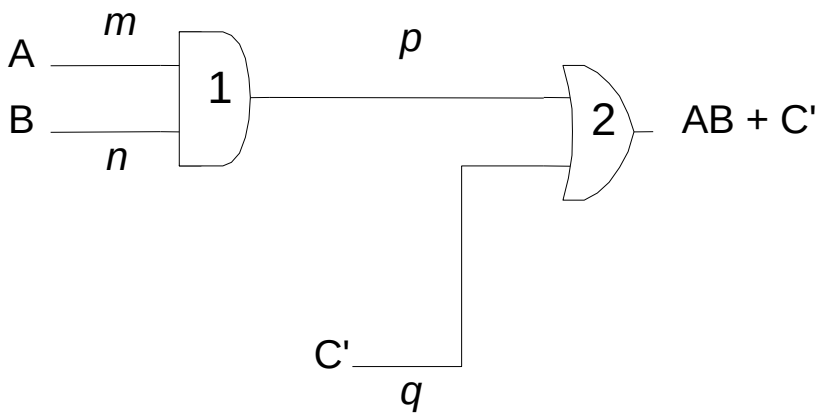


בניית טבלת התקלות

- בטבלה השלמה נחפש עמודות בהן יש *x* רק בשורה אחת, ונסמן *x*-ים אלו בעיגולים
- השורות הכוללות סימונים אלו מייצגות "בדיקות חיוניות" (essential tests)
- על מנת להרכיב ניסוי לגילוי תקלות, נוסיף לבדיקות החיוניות עוד בדיקות (שורות) כך שבכל עמודה יהיה *x* אחד לפחות
- בדוגמה לעיל שלוש בדיקות חיוניות: 011, 101, 111, וביחד הן מגלות את כל התקלות חוץ מ-q0
- בדיקה רביעית יכולה להיות 000 או 010 או 100, לפי רצוננו. ביחד,.....

Inputs	Possible Faults							
ABC	m0	n0	p0	q0	m1	n1	p1	q1
000				x				
001							x	x
010				x				
011					x		x	x
100				x				
101						x	x	x
110								
111	x	x	x					

ניסוי לגילוי תקלות



- הניסוי יורכב כלהלן :
  - הכנס כניסה 011 : אם היציאה 1, המעגל מקולקל אחרת,
  - הכנס כניסה 101 : אם היציאה 1, המעגל מקולקל אחרת,
  - הכנס כניסה 111 : אם היציאה 0, המעגל מקולקל אחרת,
  - הכנס כניסה 000 : אם היציאה 0, המעגל מקולקל אחרת, המעגל תקין

Inputs	Possible Faults							
ABC	m0	n0	p0	q0	m1	n1	p1	q1
000				x				
001							x	x
010				x				
011					x		x	x
100				x				
101						x	x	x
110								
111	x	x	x					

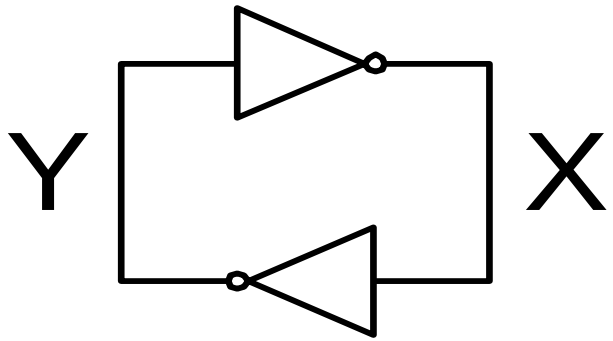
מגבלות השיטה

- לשיטה שתוארה עד כה יש שתי מגבלות:
  - הטבלה עלולה להיות גדולה מידי עבור מעגלים גדולים (אין הבטחה שלא נזדקק לכל 2^n השורות בטבלה)
  - אם נגביל את עצמנו למספר קטן של בדיקות, כיצד נבחר בניסוי שיגלה את המספר הגבוה ביותר של תקלות?
- במקרה השני, אנו מגדירים "כיסוי תקלות" (fault coverage) כאחוז התקלות שהניסוי מגלה מתוך סך כל התקלות האפשריות
  - במקרים מעשיים ניתן להגיע לאחוזים גבוהים למדי, מעל 95%

Agenda

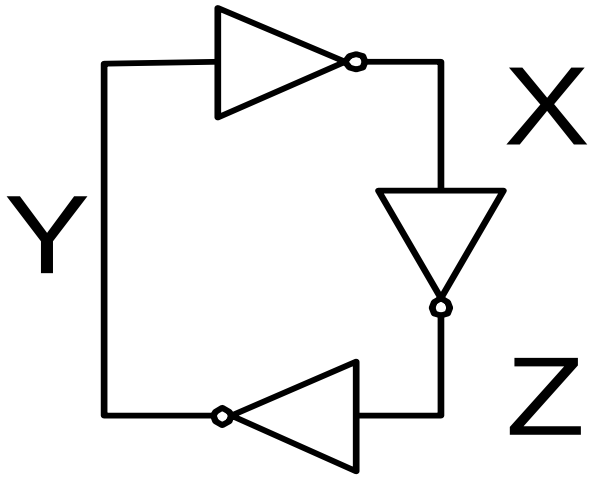
- Arithmetic logic circuits
  - Full adder
  - Ripple carry adder
  - Carry lookahead adder
  - RISC-V ALU
- Fault detection in combinational logic
- **Memories**
  - Latch, flip-flop
  - Clock
  - Metastability
  - Register
  - RISC-V register file
  - Shift register
  - Memory types

יחידות זיכרון מאפשרות מערכות סדרתיות

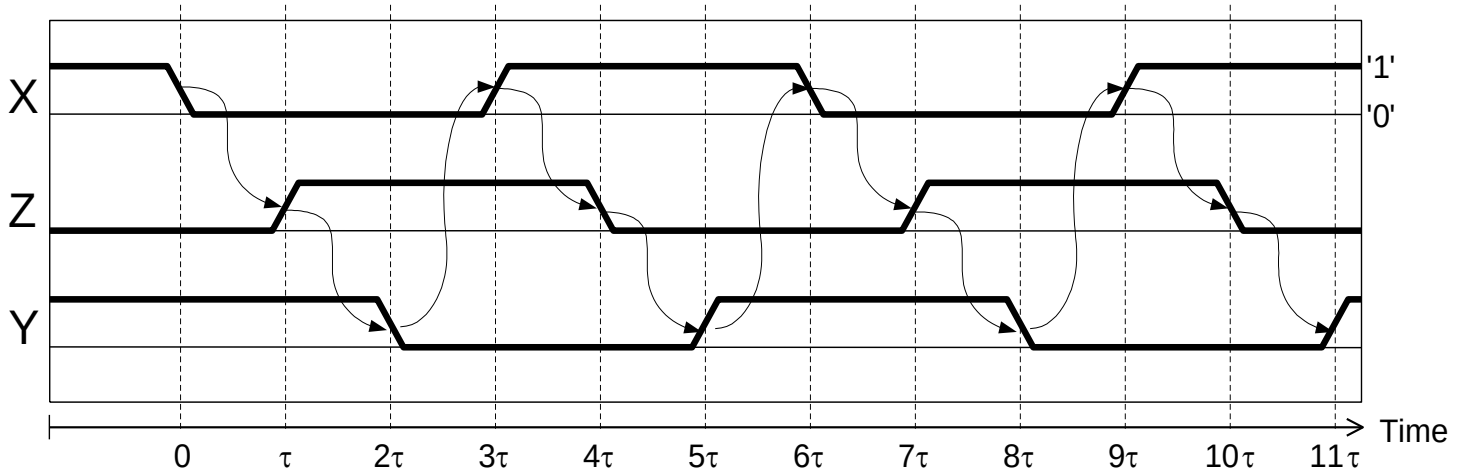


- עד כה עסקנו במערכות צירופיות
  - ערכי המוצא נקבעים לפי ערכי המבוא בלבד
  - אין מסלולים מעגליים
- כעת נבדוק מערכות בעלות **זיכרון**
  - דוגמה: שני מהפכים במעגל סגור
  - X=not(Y), Y=not(X)
  - מצב יציב: X=1, Y=0
  - מצב יציב נוסף: Y=1,X=0
- למערכת זו שני מצבים יציבים
  - כאשר היא תתייצב באחד מהם, היא תישאר בו לנצח
  - זו מערכת דו-יציבה (bi-stable) והיא ראויה לשמש בתור רכיב זיכרון

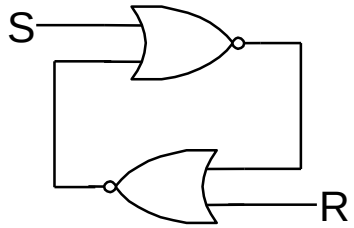
מערכת רוטטת איננה דו-יציבה

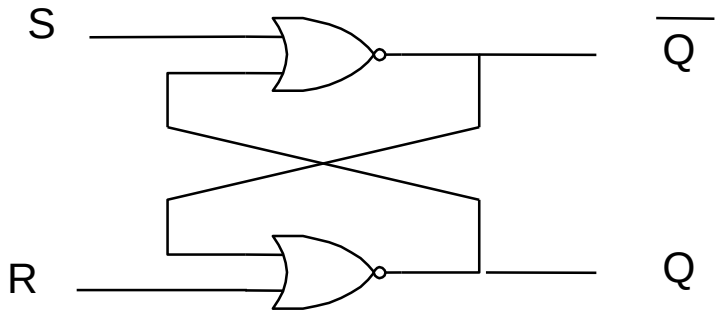


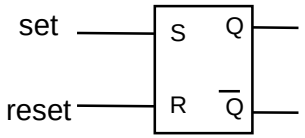
- דוגמא נוספת: שלושה מהפכים במעגל סגור
  - X=not(Y), Z=not(X), Y=not(Z)
  - Y=not(not(not(Y)))=not(Y) ←
  - זו סתירה לוגית, כלומר מצב בלתי יציב
- נניח שהשהיית כל מהפך היא τ
- המערכת רוטטת (מתנדנדת) בין ערכים לוגיים בלתי יציבים, עם מחזור באורך 6τ
- הכללה למעגלים בעלי n מהפכים:
  - כאשר n זוגי המערכת היא דו-יציבה
  - כאשר n אי זוגי היא אינה יציבה



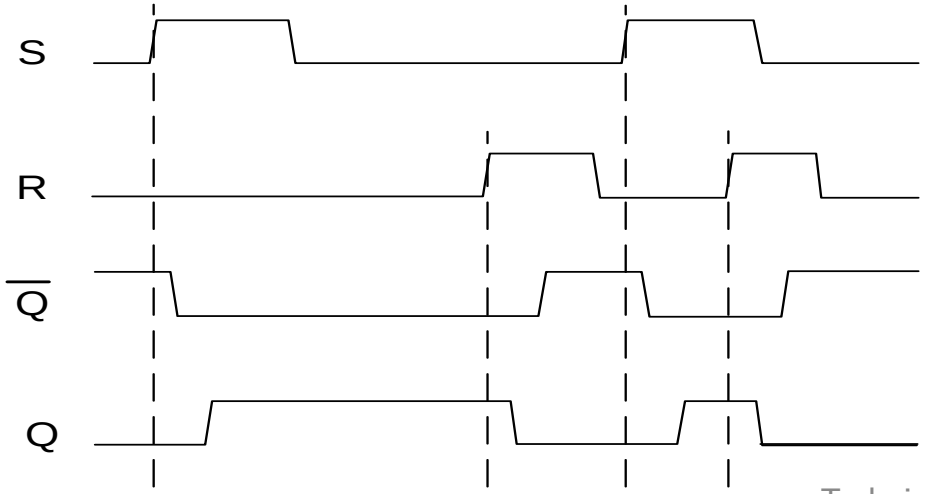
Set-Reset (SR) Latch



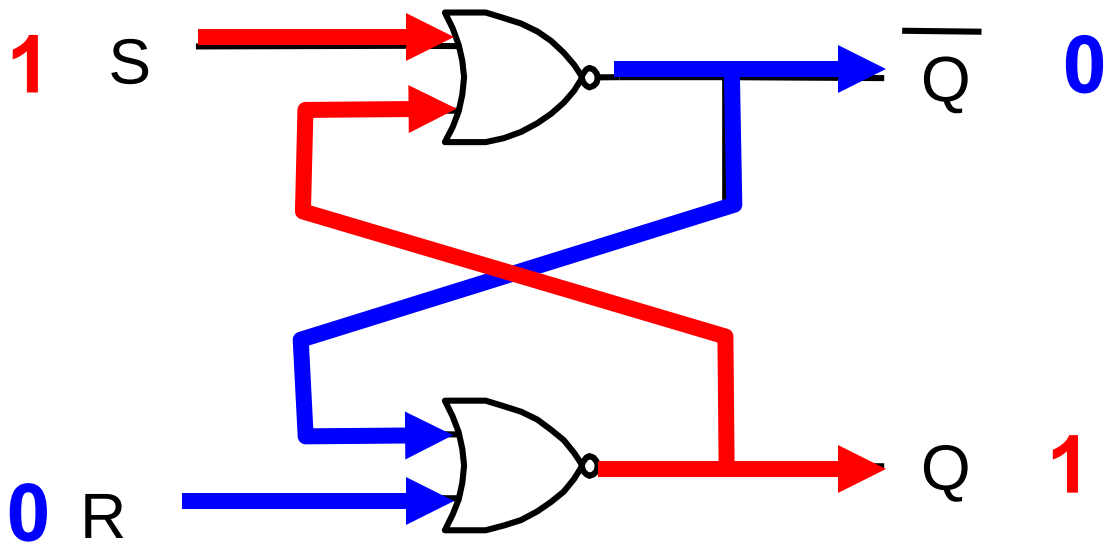




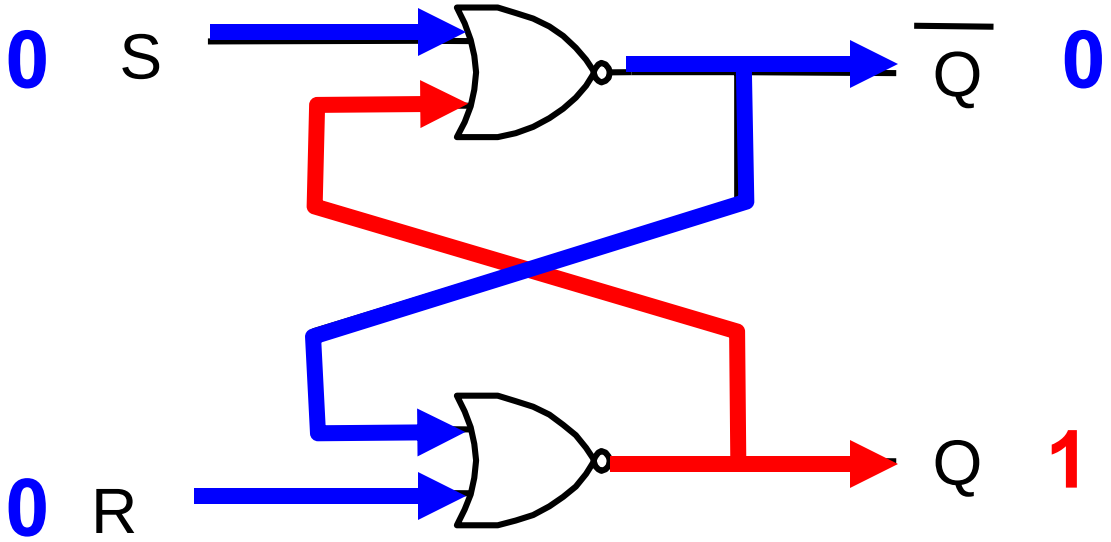
- אין דרך מעשית פשוטה לקבוע מה יהיה ערכו של זוג המהפכים הדו-יציב
- במקומם אפשר להשתמש בשערי NOR
- כאשר S=R=0 זה שקול לשני מהפכים
- כניסות S, R מאפשרות לשנות את מצבו של המעגל



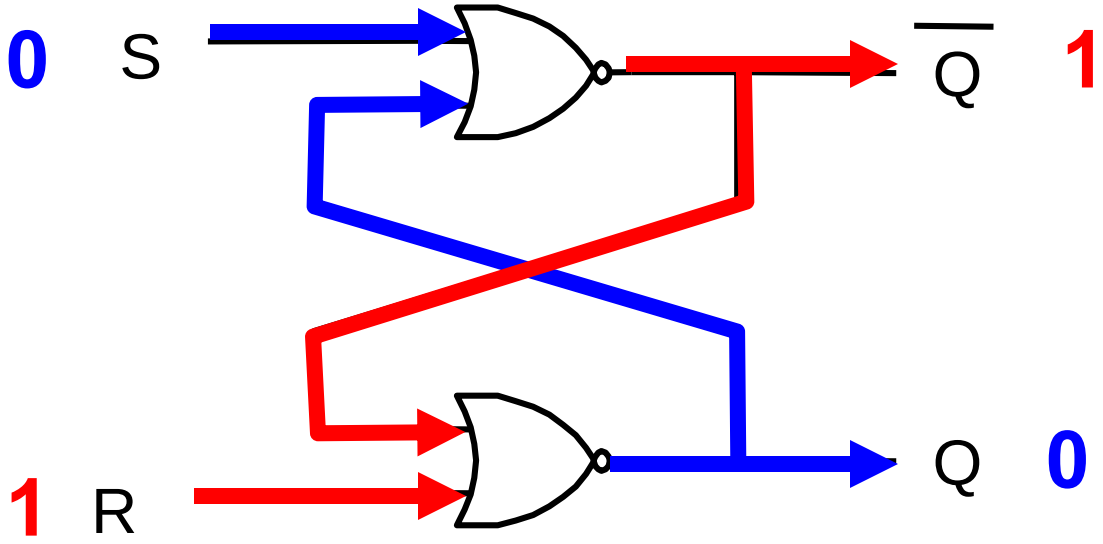
פעולת SET



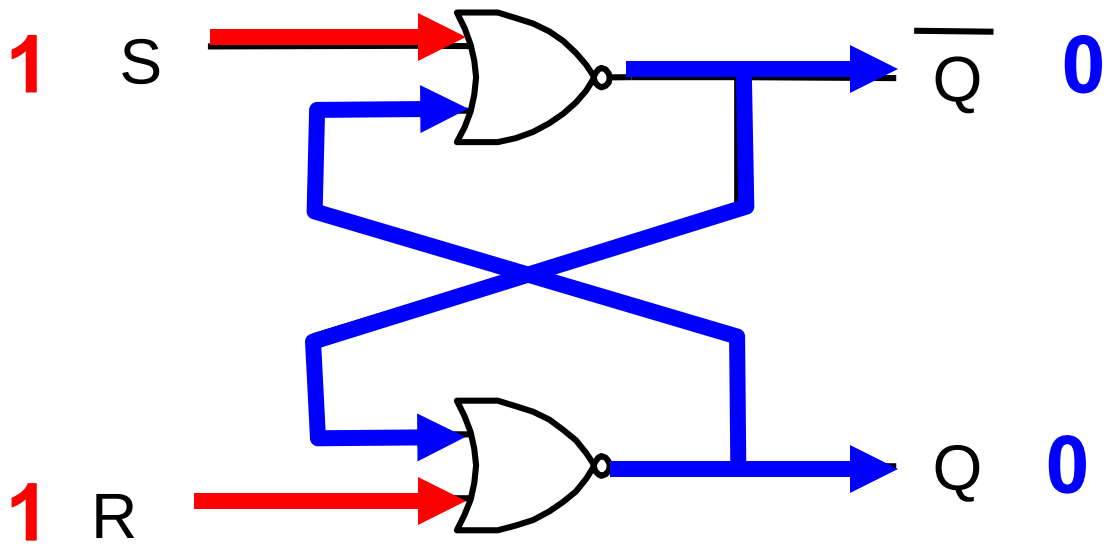
שמירת ערך



פעולת RESET

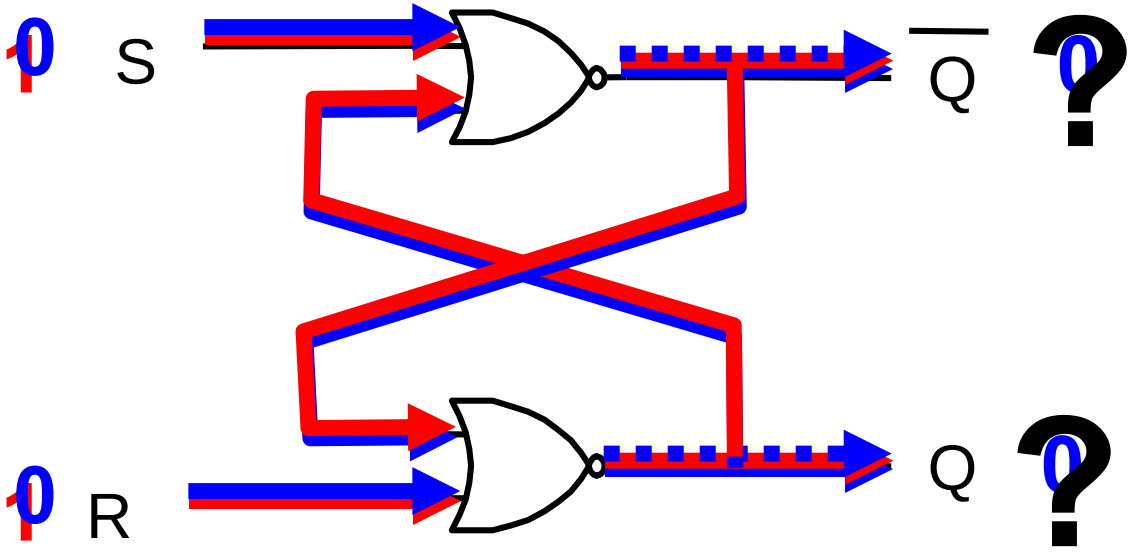


בלבול היציאות



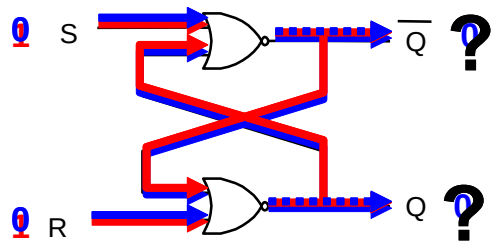
הכנסת 1 בשתי הכניסות מביאה למצב אסור ביציאות

Race !!!! מרוץ



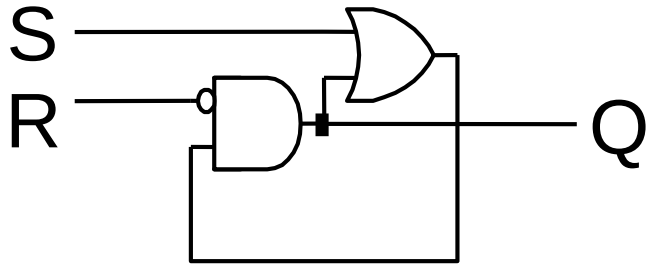
Set-Reset (SR) Latch

- אם R ו- S ירדו ביחד מ- 1 ל- 0, ינסו שני השערים להעלות את המוצא שלהם ל- '1' וייווצר מרוץ (race)
- השער המהיר יותר ינצח
- אם לשני השערים מהירות שווה בדיוק
  - יש סיכוי שהמערכת תגיע למצב "**על-יציב" (מטה-סטבילי)** שעלול להמשך זמן בלתי מוגדר
  - עד לכניסת המערכת לאחד משני המצבים היציבים (בלא יכולת לחזות מי משני המצבים יבחר)
- נדון במצב על-יציב בהמשך, ובינתיים נזכור שיש **<u>להימנע</u>** משינוי בו זמני של R ושל S מ- 1 ל- 0



Set-Reset (SR) Latch

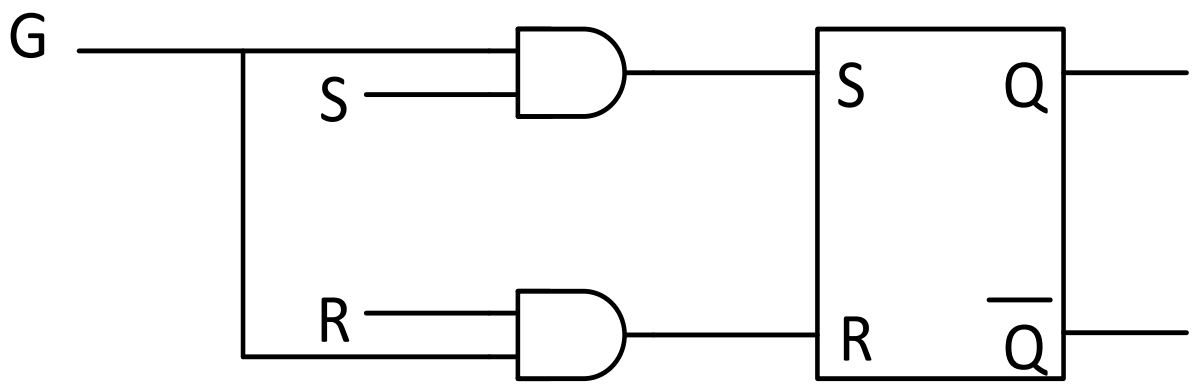
- רכיב SR זה קרוי Latch (״מנעולי״)
- שם זה שמור לרכיבי זיכרון שכניסותיהם רשאיות להשתנות בכל עת (להוציא ההגבלה הנ״ל) ושהיציאות מושפעות מיידית מן הכניסות
- דרך מקובלת אחרת לציור SR Latch היא:



- כמובן שניתן להגדיר גם latch משני שערי NAND
במקרה זה המצב נשמר כאשר S=R=1 ; ביצוע set ע״י איפוס S לזמן קצר (ואז שמירת המצב) וביצוע reset ע״י איפוס R (ואז שמירת המצב)

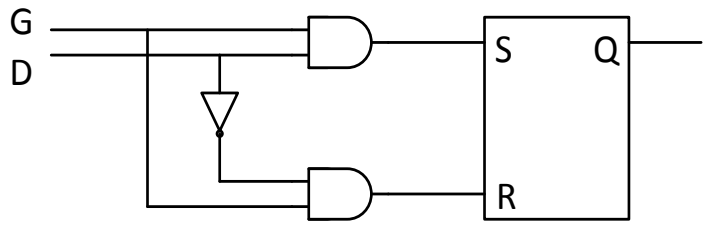
Set-Reset (SR) Gated Latch

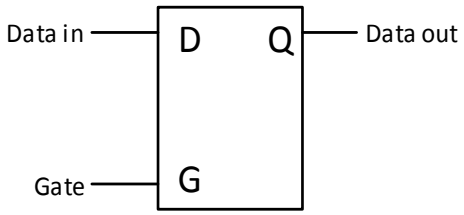
- ניתן לבקר מתי כניסות S, R תשפענה על ה- latch באמצעות המעגל הבא
- הכניסה G קרויה Gate או שער:
  - כאשר G=1 ה-latch 'פתוח' או 'שקוף'
  - כאשר G=0 ה-latch נעול ויציאותיו אינן מושפעות משינויים בכניסות
- Gated Latch נקרא גם Transparent Latch



Transparent Data (D) Latch

- ע״י שינוי קל נשתמש ב-latch לאכסון סיבית בודדת D

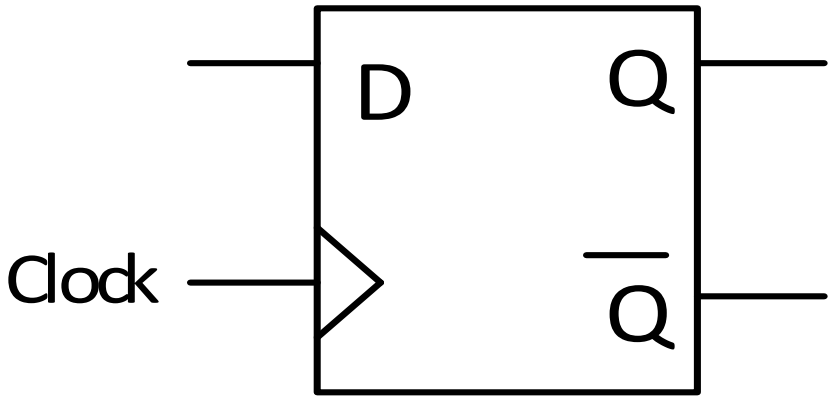




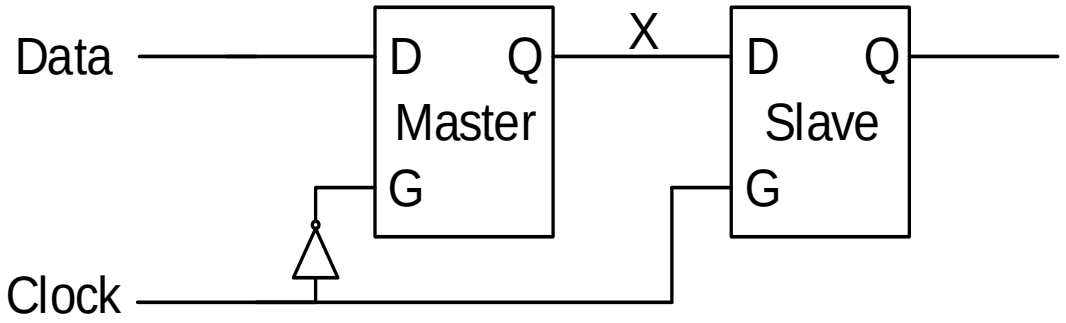
- רכיב זה נקרא D-Latch
- מקובל להכניס בכניסתה-Gate אות מחזורי הנקרא שעון (Clock)
- נדרוש שכאשר כניסה אחת משתנה, השנייה יציבה
- נדרוש שכאשר D משתנה, G=0
- תנאים אלו יבטיחו שמצב S=R=1 בלתי אפשרי

Positive Edge Triggered D Flip Flop (ET-D-FF or DFF)

- במקום לאפשר שינוי כשהשעון ברמה גבוהה,
נאפשר את שינוי היציאה רק כתוצאה מ**עליית השעון**

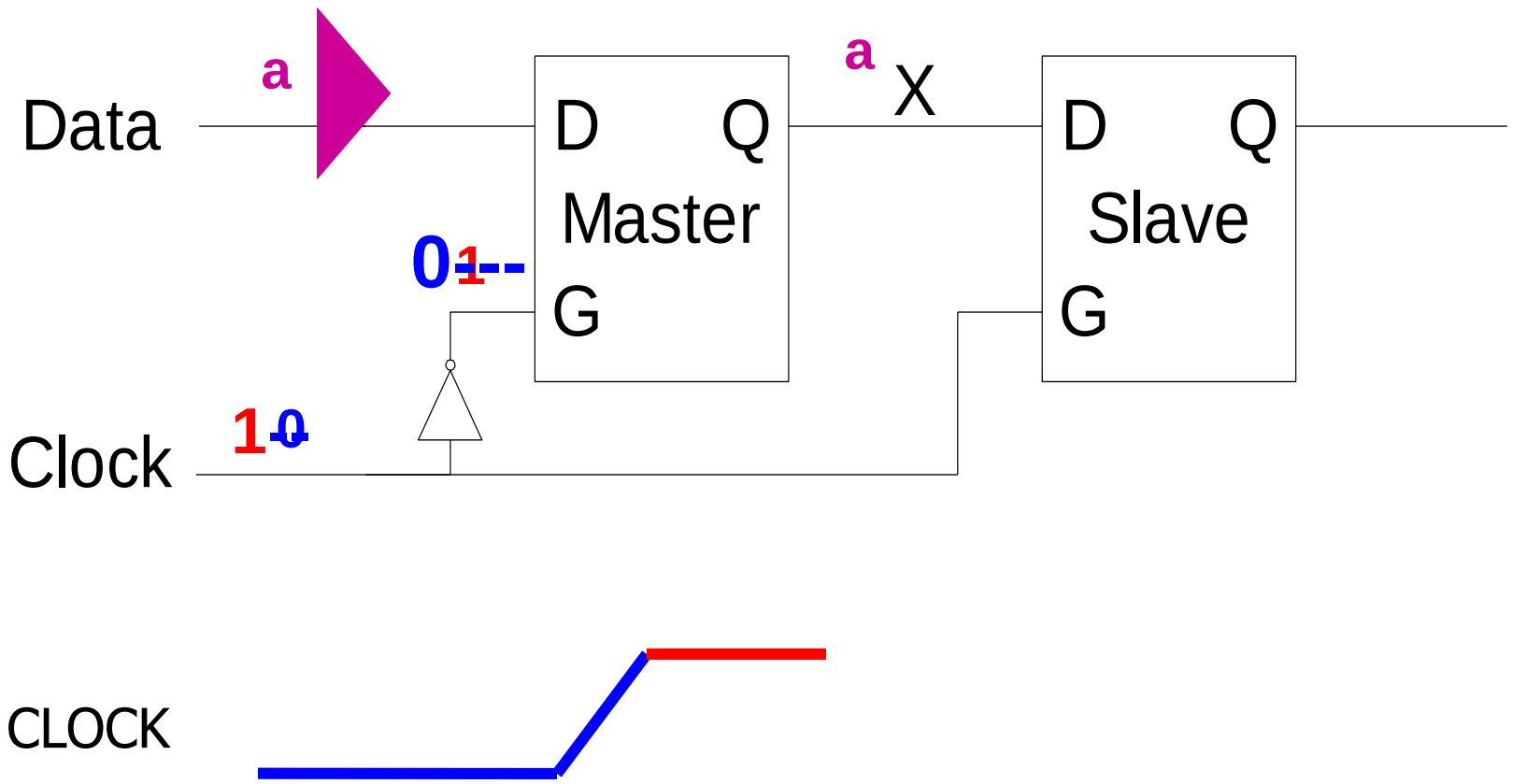


בניית DFF משני Transparent Latches

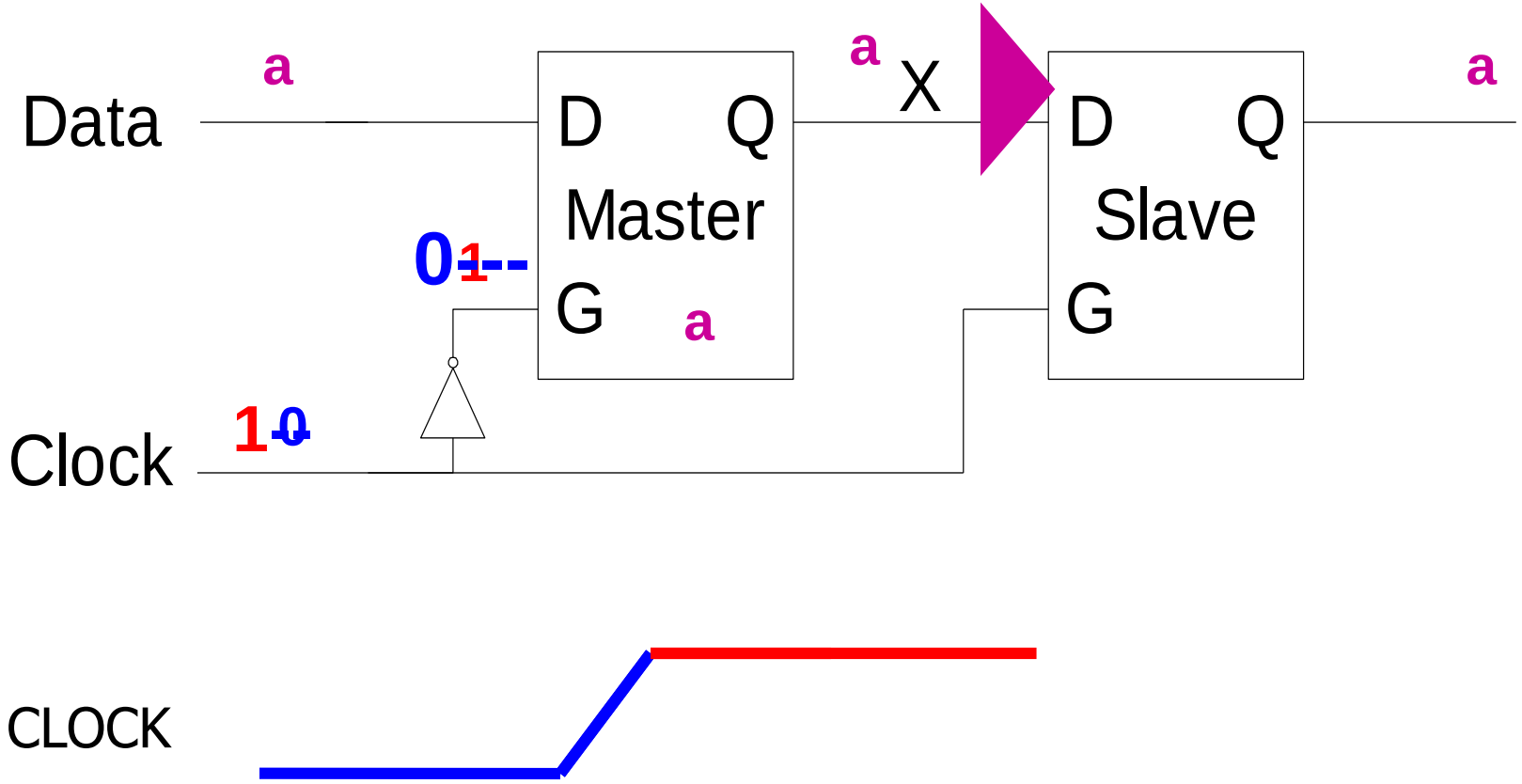


- נשתמש בשני latches
- כאשר השעון Clock=0, ה-latch הראשון (השמאלי) שקוף, וכניסת Data עוברת לצומת הפנימית X
- כאשר השעון עולה ל-'1' קורים שני הדברים הבאים:
  - ה-latch הראשון נסגר ומכאן והלאה הוא "זוכר" את ערכה האחרון של הכניסה
  - ה-latch השני נפתח וערכו של X מועבר ליציאה
- מאוחר יותר (לאחר מחצית מחזור השעון) יורד השעון שוב ל-'0', ואז קורים שני הדברים הבאים:
  - ה-latch השני נסגר ומכאן והלאה הוא "זוכר" את ערכו האחרון של X
  - ה-latch הראשון נפתח ומאפשר לכניסה Data לקבוע את ערכו הבא של X

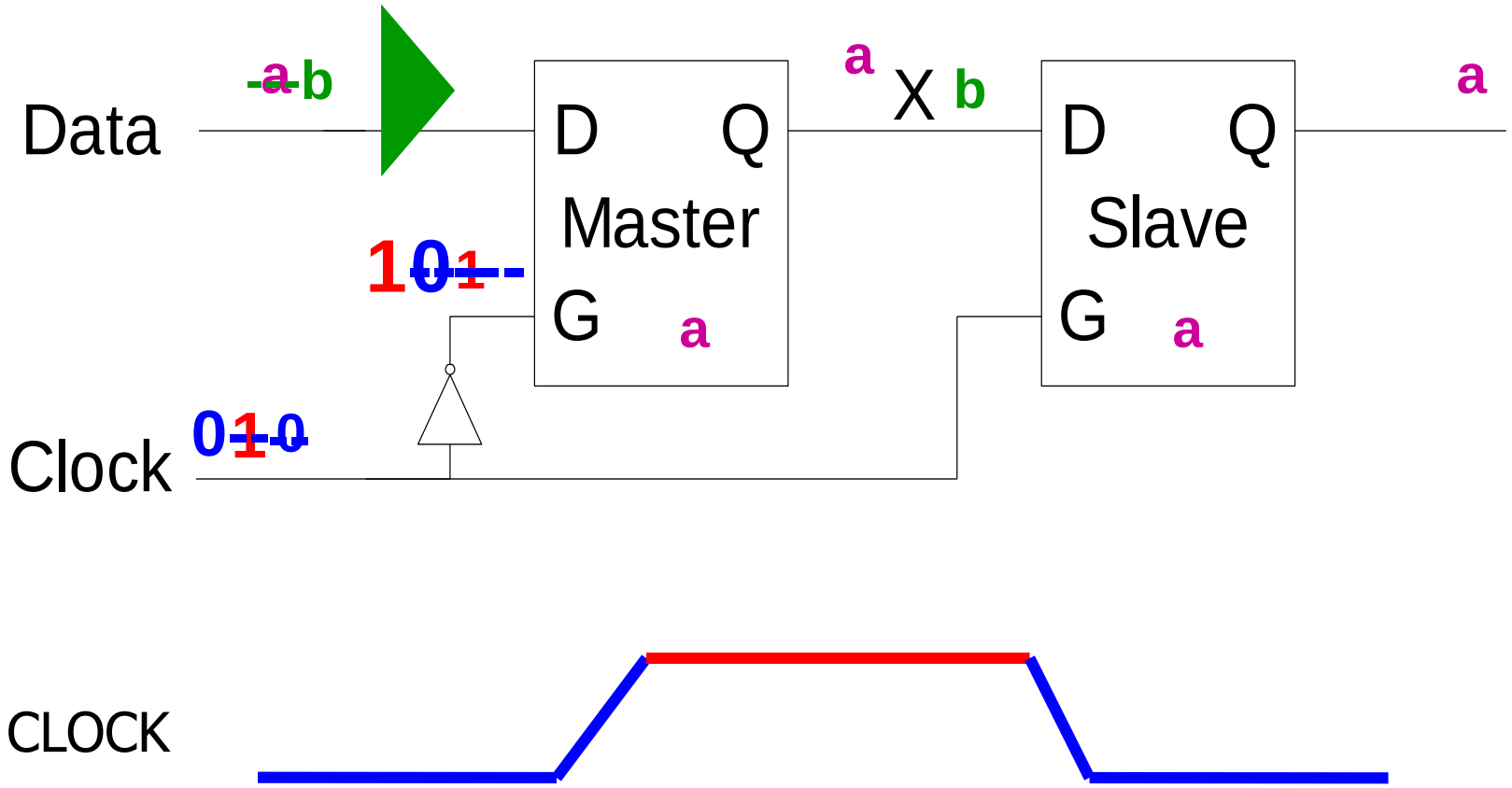
בניית DFF משני Transparent Latches



בנייתDFF משני Transparent Latches

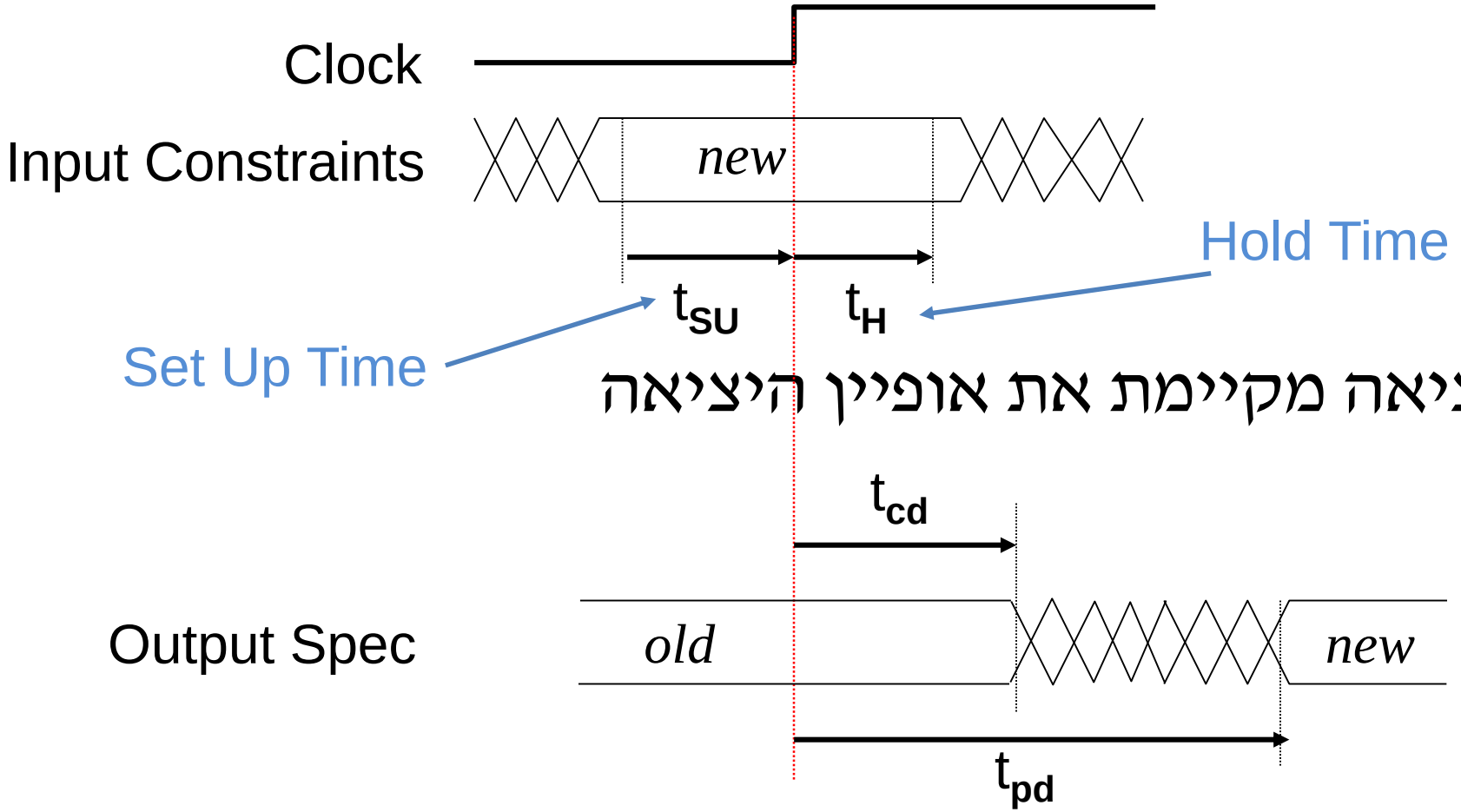


Transparent Latches בנייית DFF משני



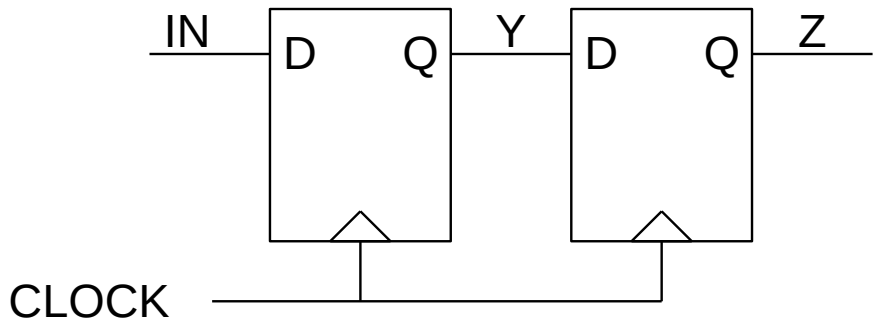
אפיון הזמנים של DFF

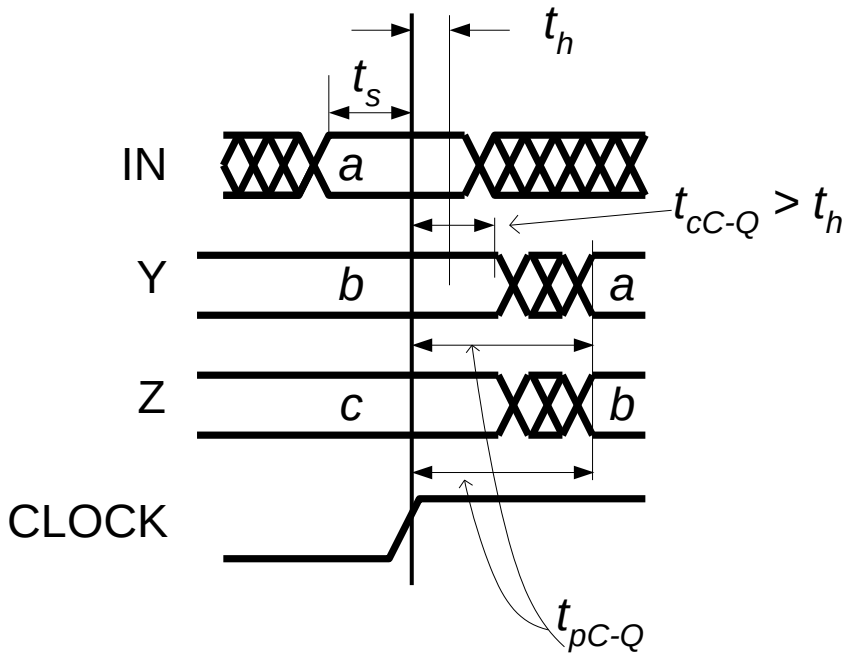
- אם הכניסות מקיימות את האילוצים (כניסה יציבה במשך t_{SU} ו- t_H)



- אזי היציאה מקיימת את אופיין היציאה

תזמונים ב-DFF מקושרים

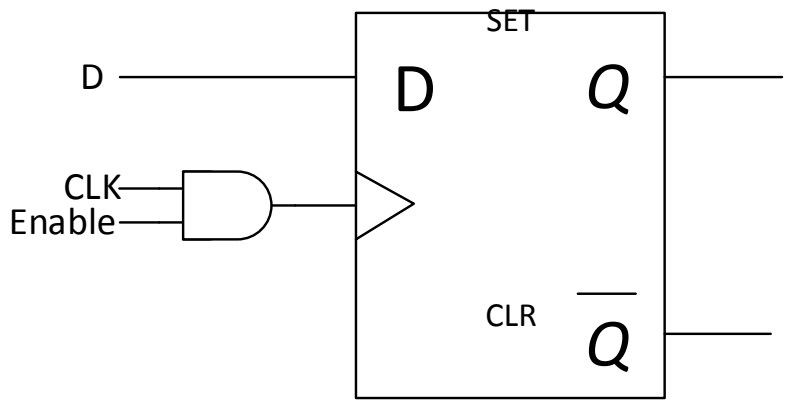


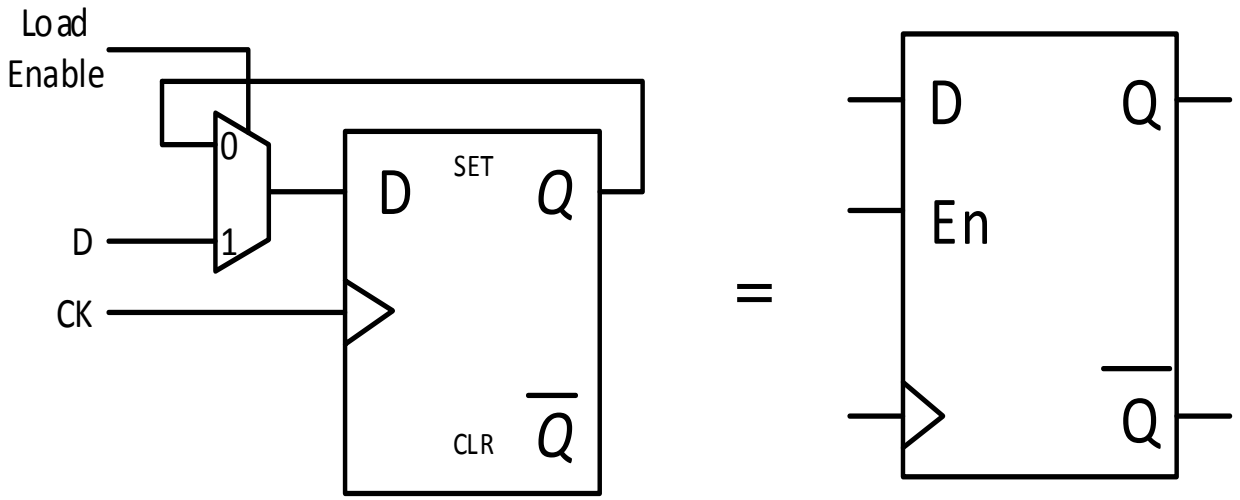


- לפני עליית השעון,
  - בכניסה IN ישנו ערך a (0 או 1)
  - ה-DFF השמאלי שומר ערך b
  - ה-DFF הימני שומר ערך c
- לאחר עליית השעון
  - ה-DFF השמאלי קיבל ושומר את a
  - ה-DFF הימני קיבל ושומר את b
  - הערך c לא נשמר יותר
- בכל עליית שעון, כל ה-DFF במעגל מעבירים את כניסותיהם ליציאותיהם

בקרת טעינה ב-DFF

- אפשר לחסום טעינת ערך חדש על ידי חסימת השעון
  - מחייב תיזמון מדויק ומסובך יחסית לשעון
- לחילופי, אות enable (או Load Enable) מאפשר לחסום את הכניסה מבלי למתג את השעון



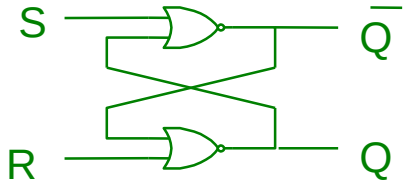


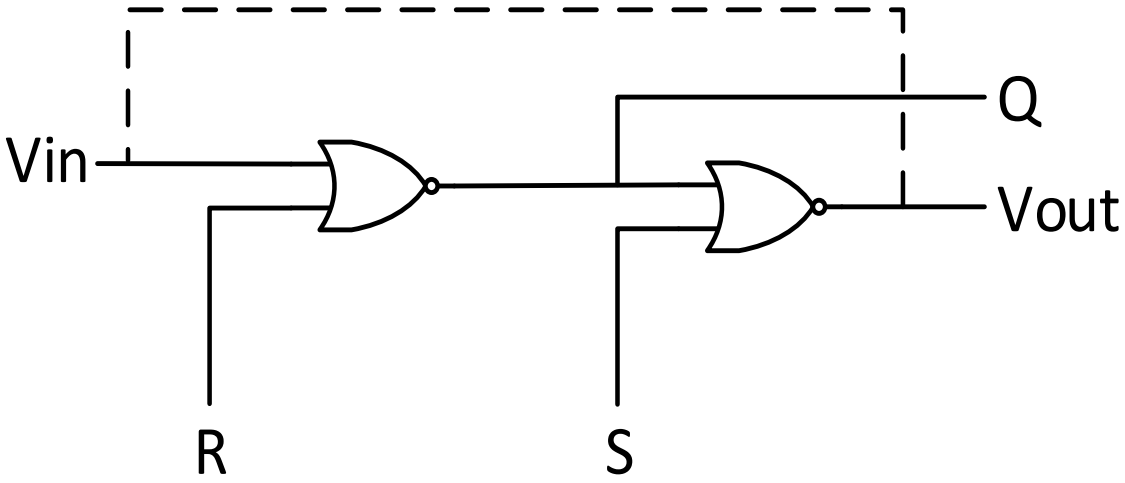
משטרי תזמון

- המערכות הספרתיות תפעלנה בהתאם לשני המשטרים להלן. המשטרים מבוססים על שני סוגי רכיבים:
  - רכיב צירופי (רכיב המתואר ע"י פונקצית מיתוג)
  - רכיב זיכרון מתוזמן (כלומר שיש לו כניסת שעון)
- המשטר הסטטי: כל רכיב צירופי מתוכנן כך שבהינתן לו ערכי כניסה לוגיים תקינים ויציבים למשך זמן מספיק, יתקבל ביציאתו ערך לוגי תקין לאחר זמן ההשהיה המתאים
- המשטר הדינאמי מחייב את קיום התנאים הבאים:
  - מעגל לוגי מורכב מרכיבים צירופיים ורכיבי זיכרון מתוזמנים בלבד
  - מעגל לוגי אינו מכיל לולאות של רכיבים צירופיים. כל לולאה חייבת להכיל לפחות רכיב זיכרון מתוזמן אחד
  - מחזור השעון ארוך מספיק על מנת לספק את דרישות הזמן של כל הרכיבים
  - בכל הכניסות, הרמות הלוגיות תהיינה יציבות למשך זמן מספיק

מטה-סטבילית (Meta-Stability)

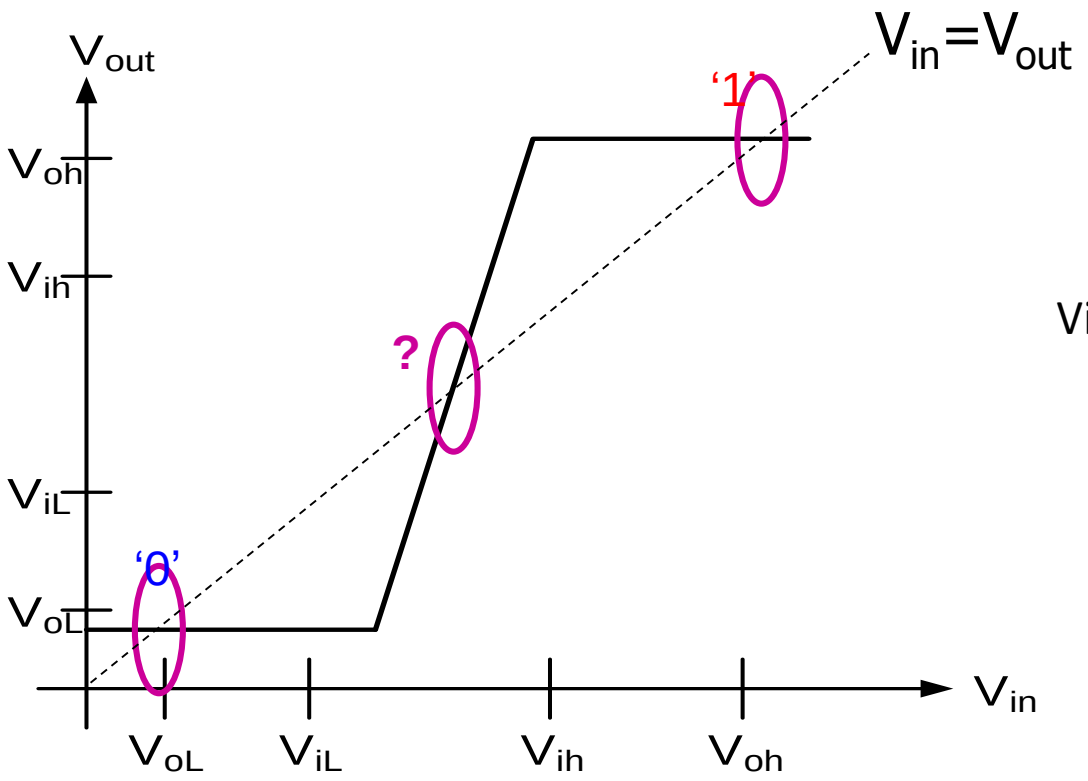
- בכדי להבין מדוע ההקפדה על המשטר הדינמי חשובה, נחזור לרגע אל SR Latch
- ללא המשוב נראה התקן זה כך:
  - המשוב מסומן בקו מקווקו

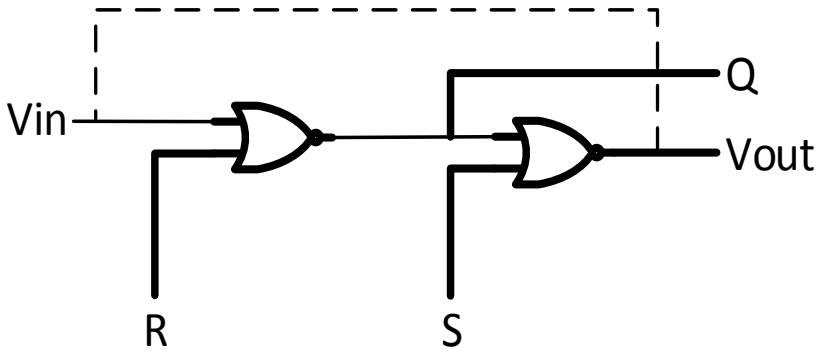




מטה-סטביליות (Meta-Stability)

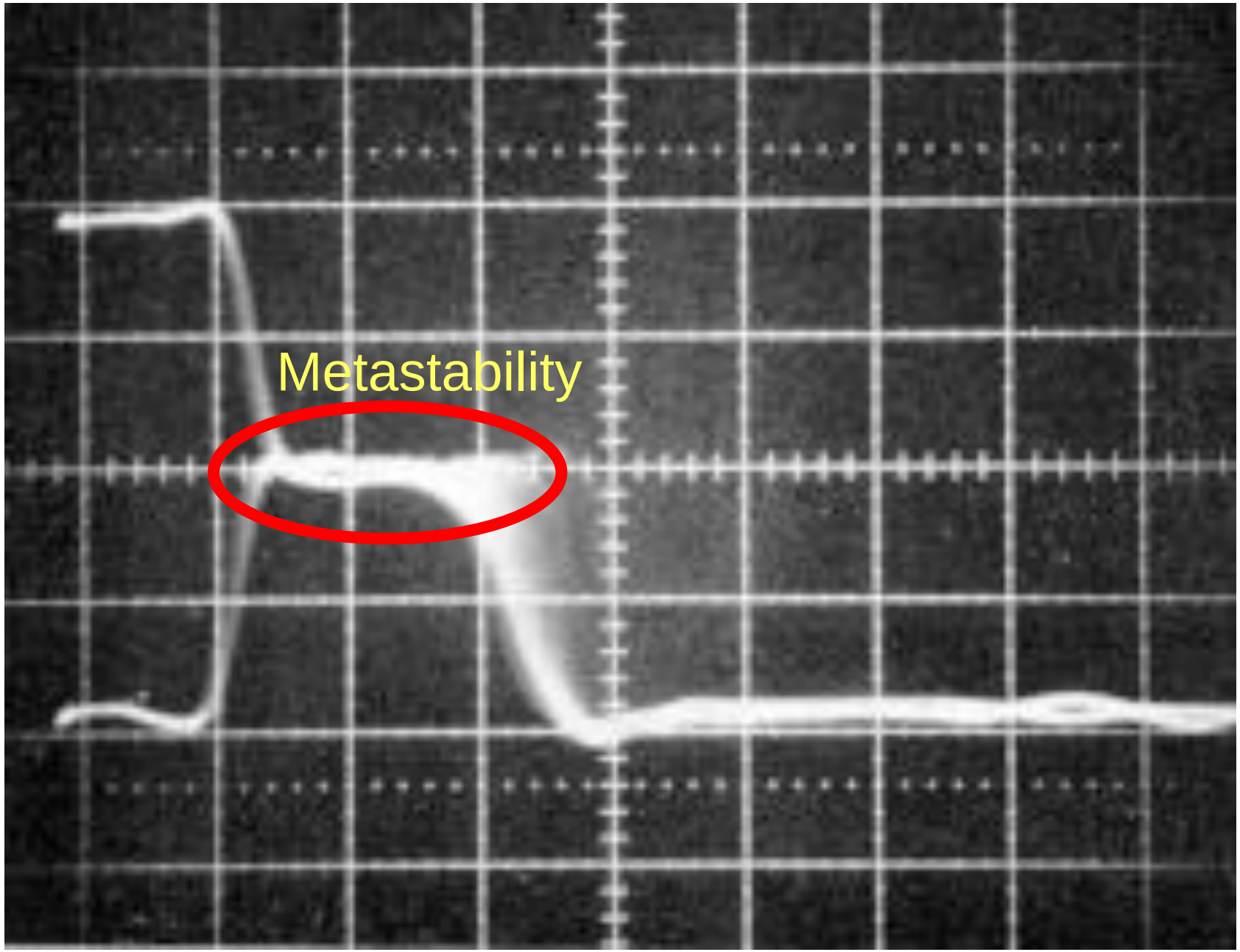
עם אפיון מתח:





מטה-סטביליות (Meta-Stability)

- אם נחזיר את המשוב, נוסף האילוץ Vout=Vin
- נקודות החיתוך בין שני הקווים מייצגות מתחים המקיימים את כל דרישות המערכת, והן לכן נקודות **שווי משקל**
- שתיים מהן מייצגות ערכים חוקיים, והן נקודות שווי משקל יציב. שינויים קטנים בערכי Vout או Vin סביבן (רעש, למשל) יוחזרו על ידי ההתקן לכוון נקודת שווי המשקל
- הנקודה השלישית איננה מנובאת על ידי המשוואות הלוגיות שכתבנו, כי היא מערבת מתחים שאינם ערכים לוגיים חוקיים. נקודה זו היא נקודת שווי משקל **רופף** – שינוי קטן בערכי Vout או Vin יגרום למעבר המערכת אל אחת משתי הנקודות היציבות
- שיווי משקל רופף זה קרוי גם שיווי משקל "על יציב" או מטה-סטבילי (meta-stable)

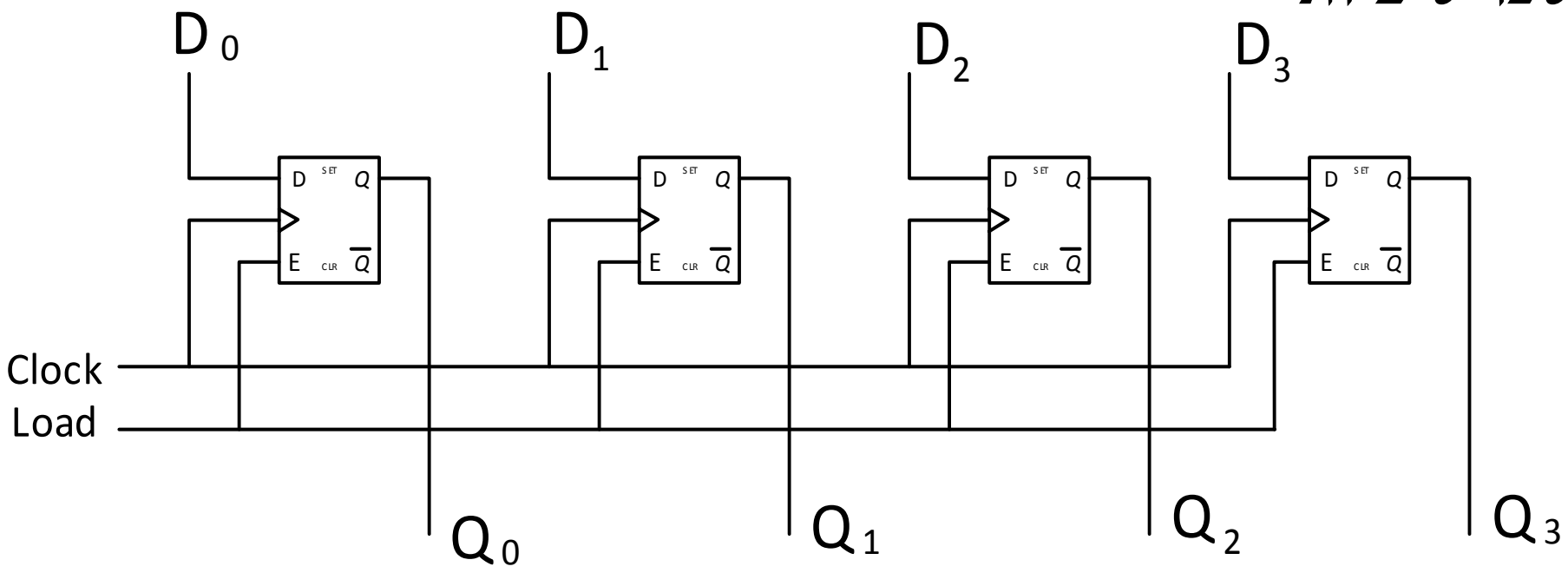

Metastability

Agenda

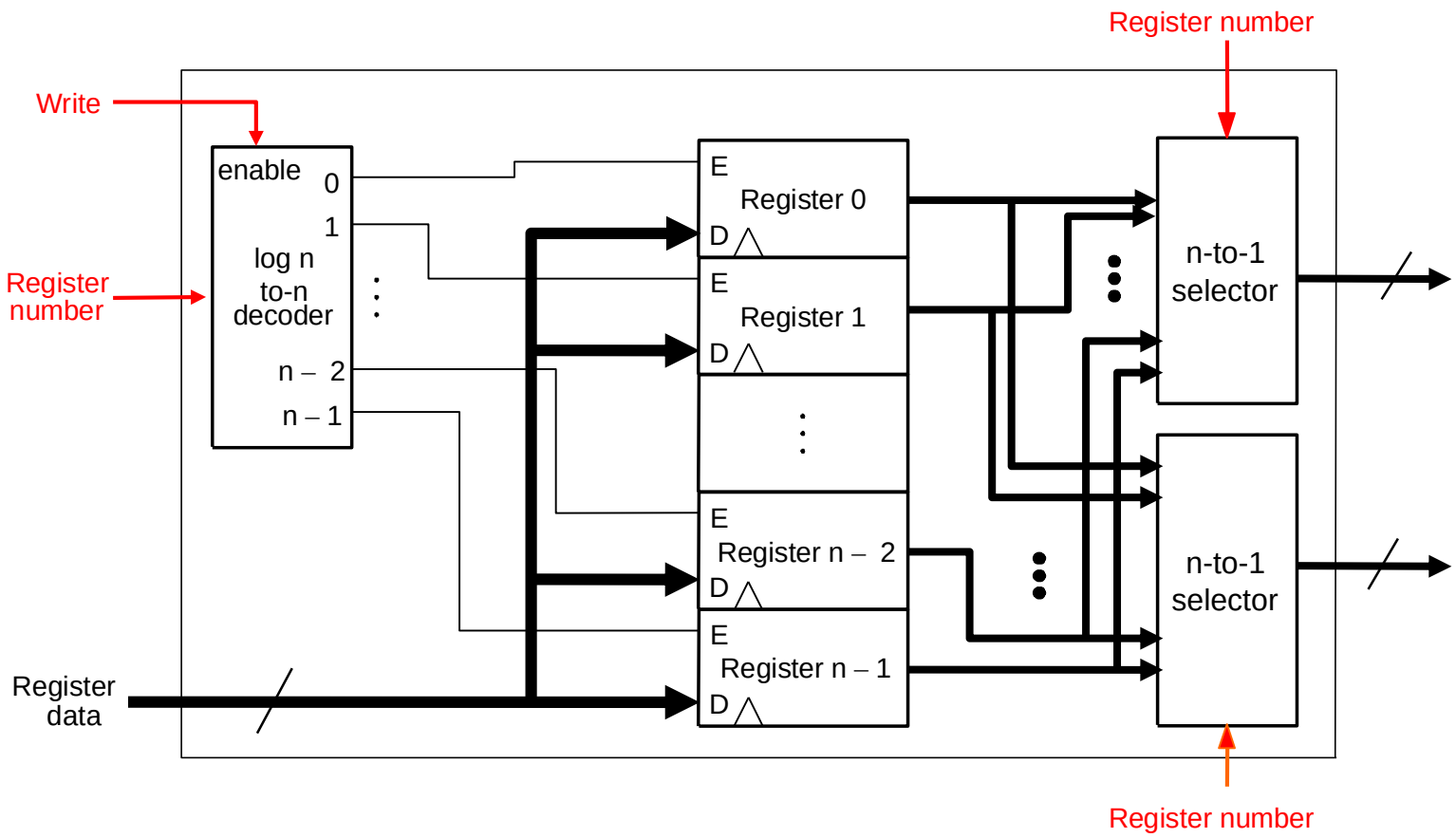
- Arithmetic logic circuits
  - Full adder
  - Ripple carry adder
  - Carry lookahead adder
  - RISC-V ALU
- Fault detection in combinational logic
- Memories
  - Latch, flip-flop
  - Clock
  - Metastability
  - Register
  - RISC-V register file
  - Shift register
  - Memory types

אוגר – רגיסטר Register

- DFF משמש לבניית אוגרים (רגיסטרים) המאפשרים כתיבה או קריאה בו זמנית של מספר סיביות

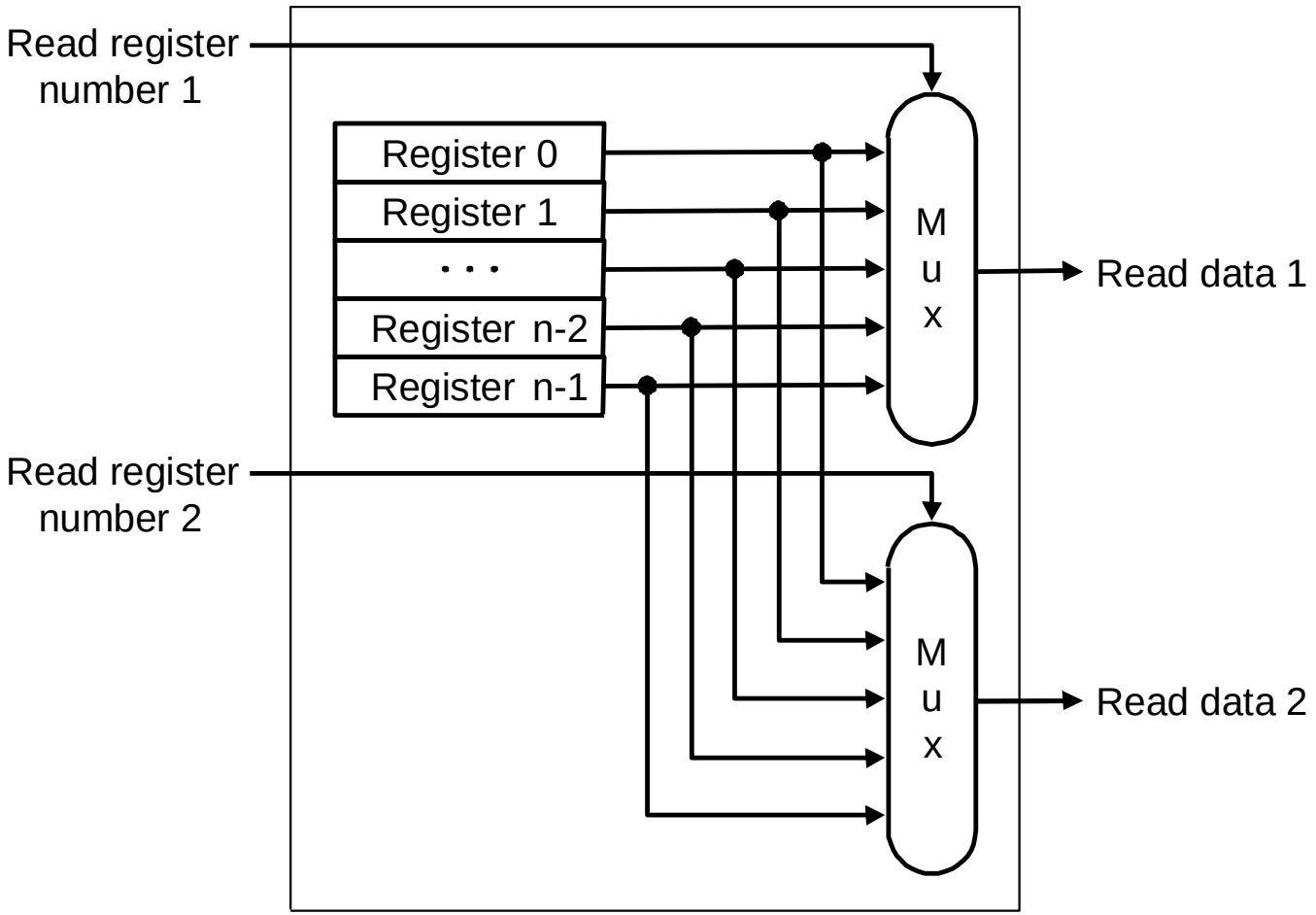


A General Register File

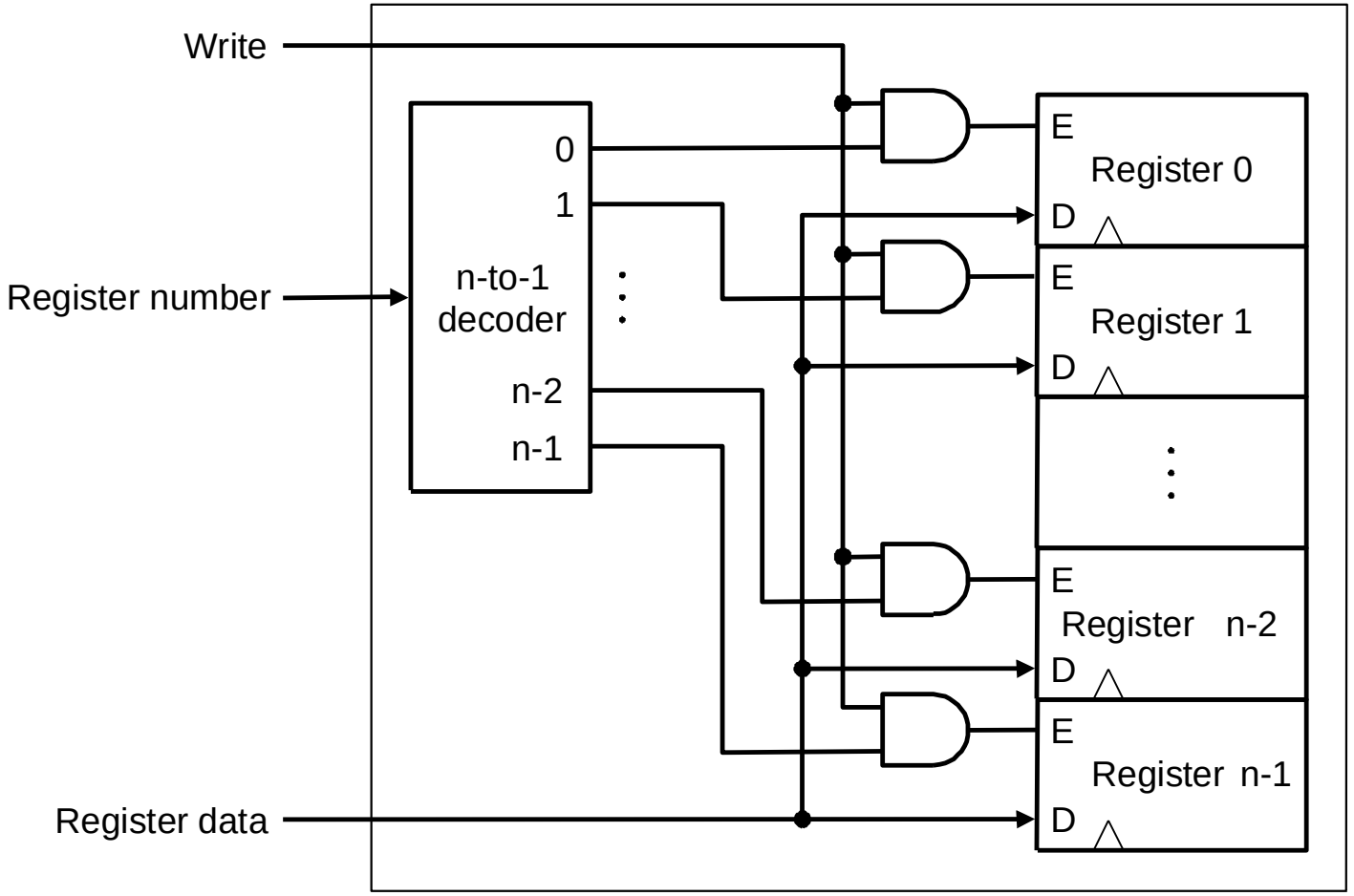


- מספר רגיסטרים
- בחירה (מפענח) באיזה רגיסטר לכתוב
- בחירה (בורר) מאיזה רגיסטר לקרוא
- אפשר לקרוא משני רגיסטרים בו-זמנית
  - מתאים למחשב

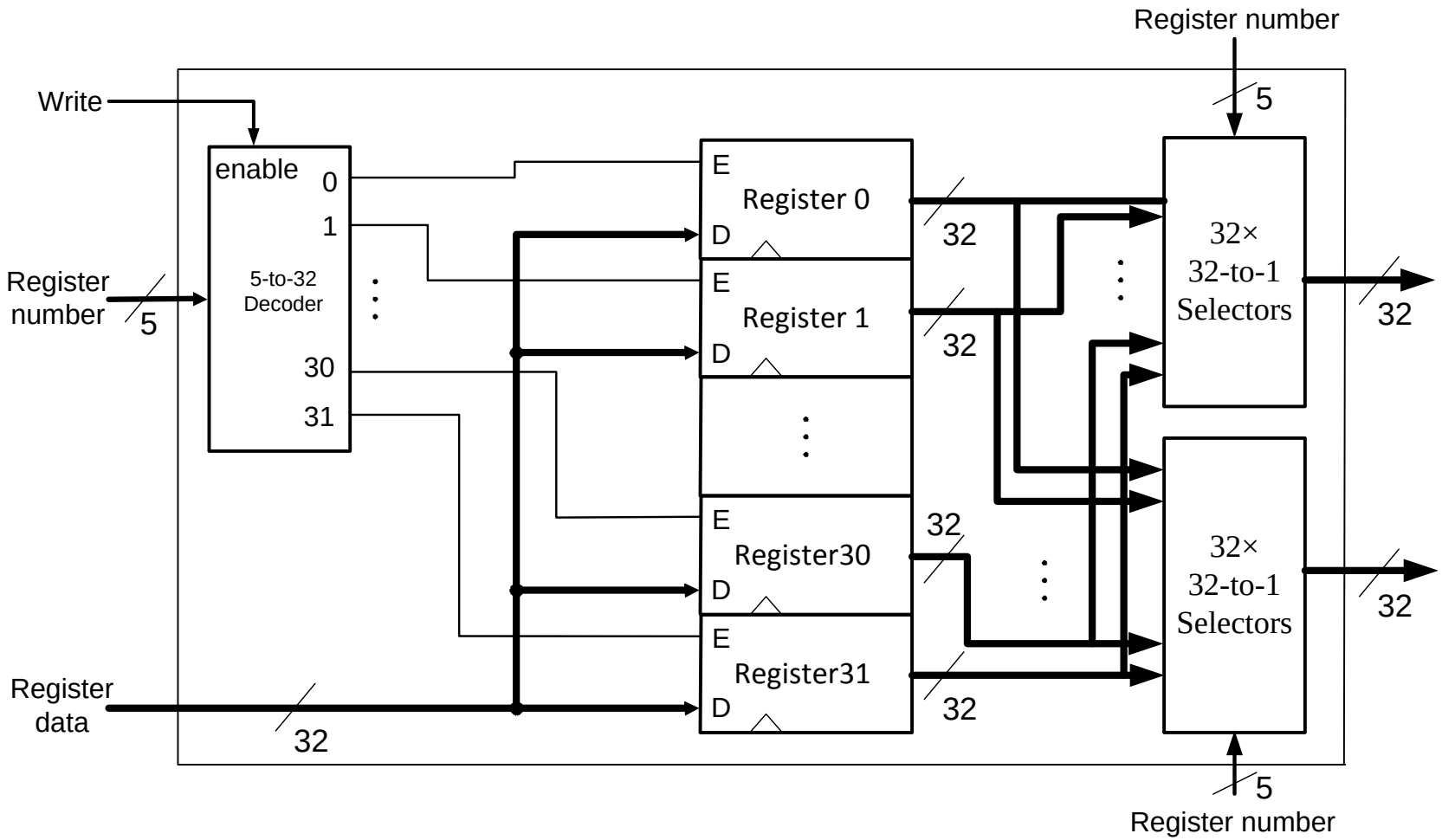
Register Files (Read Ports)

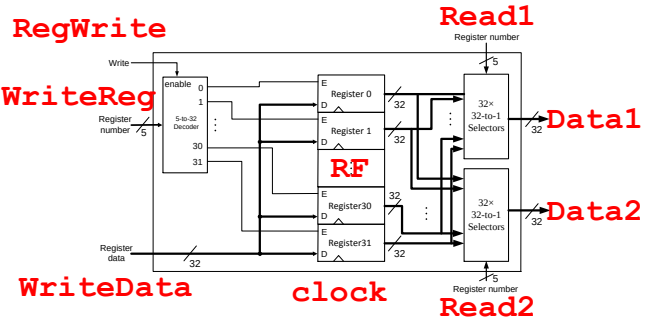


Register Files (Write Port)



RISC-V (RV32) Register File

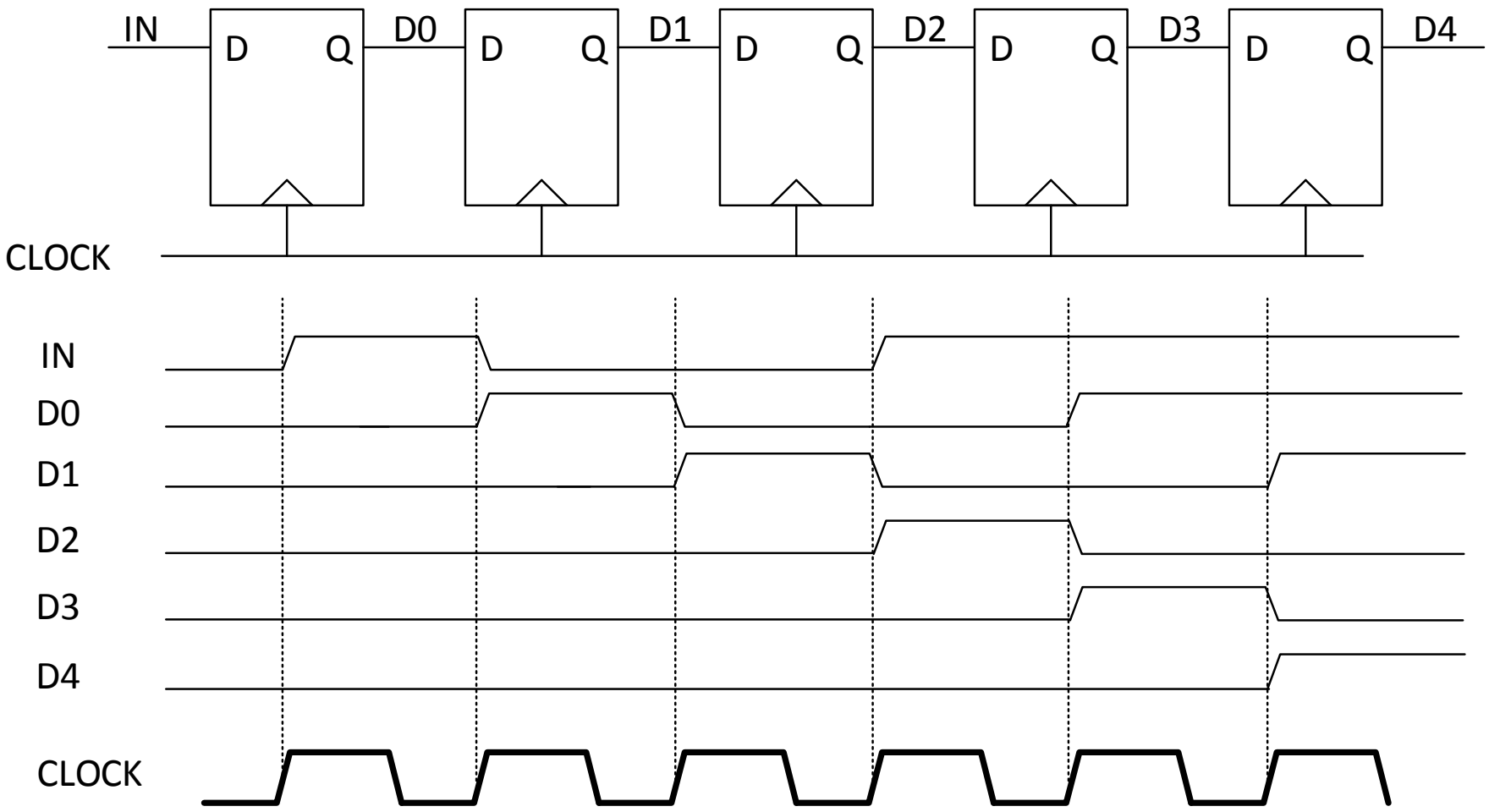




RISC-V (RV32) Register File

```
module registerfile (Read1,Read2,WriteReg,WriteData,RegWrite,Data1,Data2,clock);
    input [5:0] Read1,Read2,WriteReg; // reg numbers to read or write
    input [31:0] WriteData;           // data to write
    input RegWrite ,                  // the write control
        clock;                        // the clock to trigger write
    output [31:0] Data1,Data2;        // the register values read
    reg [31:0] RF [31:0]; // 32 registers each  32 bits long
    assign Data1 = RF[Read1];
    assign Data2 = RF[Read2];
    always begin
            // write the register with new value if Regwrite is high
        @(posedge clock) if (RegWrite) RF[WriteReg] <= WriteData;
    end
endmodule
```

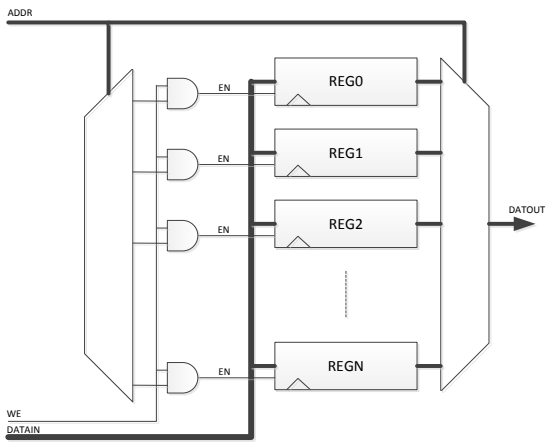
Shift Register - אוגר הזזה

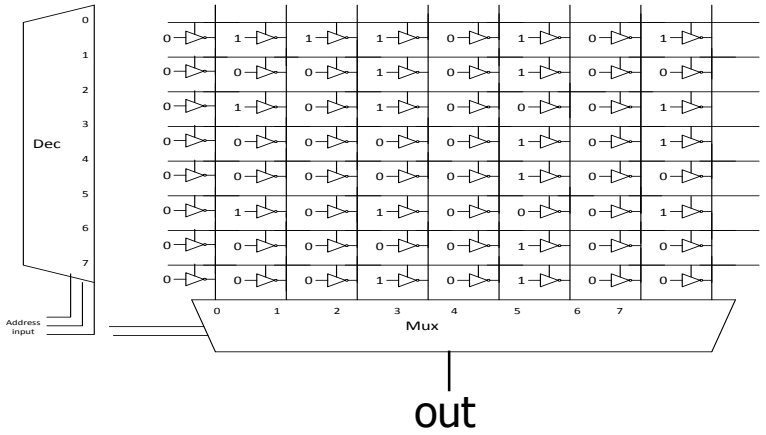


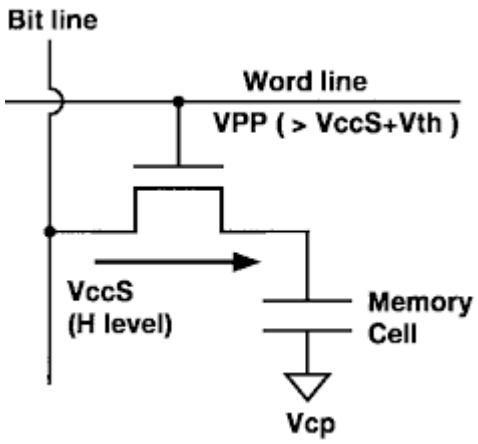
- כבר ראינו DFF קשורים
- כל סיבית מתקדמת מקום אחד בדיוק בכל מחזור שעון

Memory Types

- Register
  - Speed: Fast, Size: Large (cannot build a dense multi-bit structure), volatile
- ROM
  - Speed: Slower than flip-flop, read-only, Size: very dense (can contain many bits), non-volatile
- RAM
  - Speed: Slower than ROM, write&read, Size: Very dense (can contain many bits), volatile

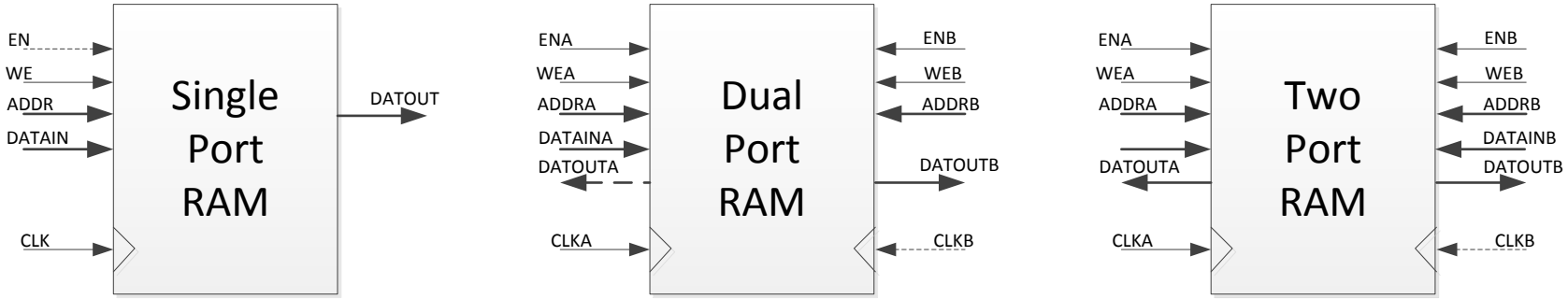


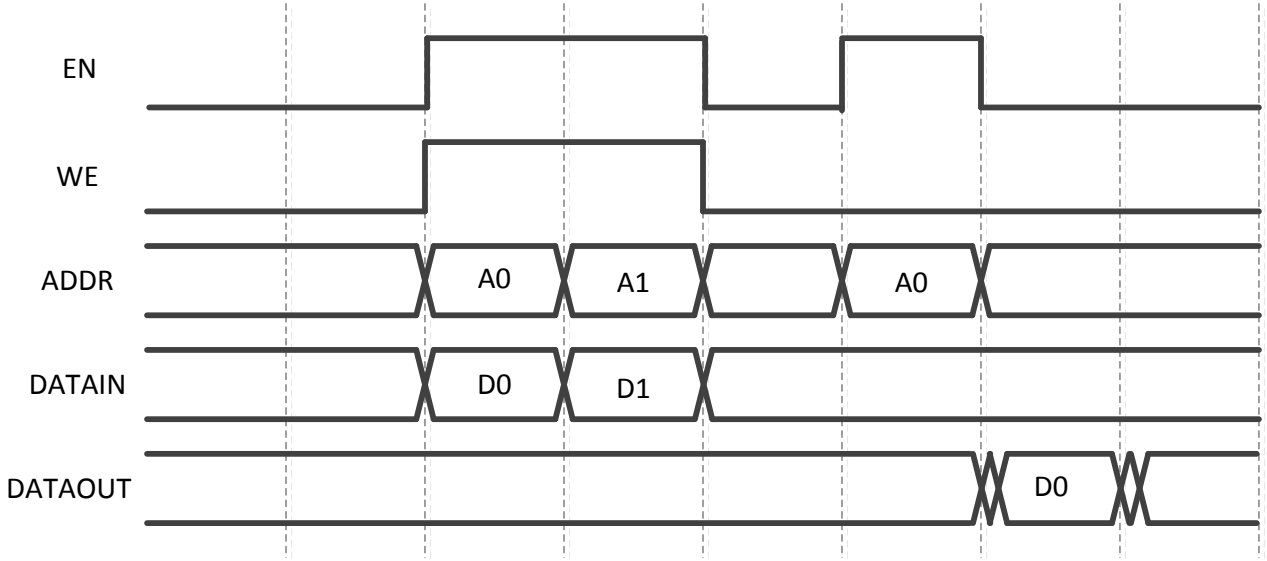




Memory Types & Interfaces

- Single, Dual and Two Port RAMs





Summary

- Arithmetic logic circuits
  - Full adder
  - Ripple carry adder
  - Carry lookahead adder
  - RISC-V ALU
- Fault detection in combinational logic
- Memories
  - Latch, flip-flop
  - Clock
  - Metastability
  - Register
  - Shift register
  - RISC-V register file
  - Memory types