מבנה המחשב | 67200

הרצאות | אוהד פאליד ורון גבור

כתיבה | נמרוד רק

תשפ"ג סמסטר ב'

תוכן העניינים

שבוע I | מבוא ומוליכים-למחצה

הרצאה

המחשבים הראשונים היו עצומים, כבדים ויקרות, ויחידת הבסיס של המעבד שלהן היה שפופרות קטודיות (אבי הטרנזיסטור) ואז שפופרות ריק, וכיום משתמשם בטכנולוגיית CMOS שמאפשרת גדילה בעשרות סדרי גודל בזמן המחזור, מהירות השעון, מספר הטרנזיסטורים ועוד. הקורס עוסק במבנה המעבד בעיקר.

בתוך כל מחשב יש אביזרי קלט ופלט (חיישני סונאר, מסך, עכבר), אמצעי אחסון (נדיף כמו RAM ולא נדיף כמו דיסק קשיח), מעבד, מערכת הפעלה, דרייברים ותוכנה. בקורס נעסוק במעבדים ותוכנה ונזכיר מערכות הפעלה ואמצעי אחסון.

קצב ההתקדמות עד לשנים האחרונות התנהג לפי חוק מור (1965) - כל שנה מצליחים להכפיל את מספר הטרנזיסטורים כל שנתיים. העליה במספר הטרנזיסטורים מספקת גם עלייה אקספ' בביצועים, ביעילות אנרגיה וגם בגודל האחסון שאפשר לייצר.

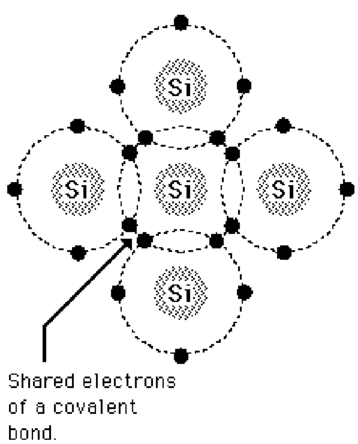
רקע כימי לטרנזיסטורים

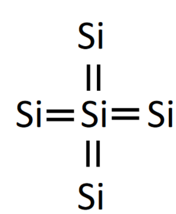
המודל של בוהר ניסה להסביר את תופעת התכונות המשותפות ליסודות באותה עמודה של הטבלה המחזורית אותה בנה מנדלייב כמה עשרות שנים קודם לכן. המודל קובע שכל חומר מורכב מאטומים, שלהם יש גרעין עם פרוטונים (חלקיקים עם מטען חשמלי חיובי) וניוטרונים (חלקיקים ללא מטען) ואלקטרונים (עם מטען שלילי) שסובבים את הגרעין.

דוגמה סיליקון (צורן) הוא מספר 14 בטבלה המחזורית, כלומר יש לו 14 פרוטונים (ובמקרה הזה גם 14 ניוטרונים) וענן אלקטרונים בשכבות שונות סביב הגרעין עם מספר שונה של אלקטרונים בכל שכבה.

בוהר גילה בנוסף שהמסלול (המעגל, השכבה של אלקטרונים) האחרון של כל אטום במצב יציב הוא מלא, לכן אטום ירצה לתת או לקחת אלקטרונים מאטומים אחרים כדי לקבל מסלול מלא.

דוגמה הרבה אטומים של סיליקון יוצרים יחד במצב יציב גביש שמורכב משריג אטומי סיליקון עם מסלול אחד מלא לכולם כך שהם חולקים אלקטרונים (ראו איור במקרה של חמישה אטומים)





קשר בו אטומים חולקים (sharing) אלקטרונים נקרא קשר קוולנטי (covalent bond).

יון הוא אטום או מולקולה עם מספר לא שווה של פרוטונים ($+$) ואלקטרונים ($-$) והמטען של החלקיק הוא הפרש הערכים האלו.

קשר יוני הוא קשר בין אטומים שנוצר כשאלקטרון עובר מהמסלול האחרון של אטום אחד לאחר (כדי להשלים מסלול) אבל מספר הפרוטונים באטומים שווה למספר האלקטרונים בכל אטום לפני מעבר האלקטרון. כך לשניהם כעת יש מטען מנוגד לשני ומשם ניגודיות המטענים מקרבת (באמצעות כוח משיכה אלקטרן-סטטי) בין האטומים לכדי קשר.

רקע פיזיקלי לטרנזיסטורים

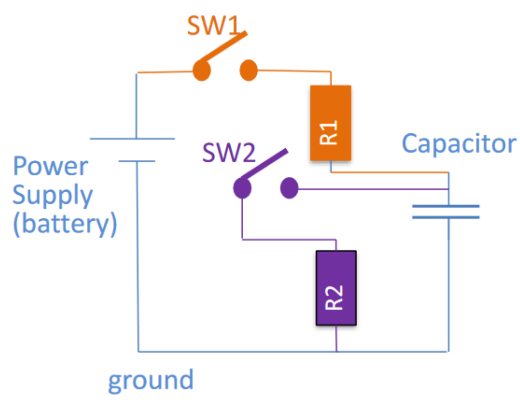
זרם חשמלי הוא תנועה של אלקטרונים (או באנלוגיה תנועה של חורים, כאשר האלקטרונים השליליים עוברים בין חורים שהם אחרת חיוביים). בקונבנציה הזרם נע מה-$+$ ל-$-$.

מתח (פוטנציאל) נמדד בוולט והוא היכולת להזרים, כאשר סוללה בעצם פשוט מחזיקה מתח וכמו מגדל מים באנלוגיה מים, אפשר לעשות בו חור וכך לגרום לזרימה אבל כל עוד לא מחברים/מחוררים את הכלי לא יקרה שום דבר.

מטען הוא מספר האלקטרונים שמאוחסנים.

קיבול זו היכולת להחזיק מטען, כאשר בעת אחסון מטען נוצר מתח בקבל (מקביל למיכל מים ולחץ).

דוגמה נביט במעגל הבא. R1 הוא נגד (שקול לצינור צר יותר, שיוצר התנגדות). אם נסגור את המתג הראשון, נסגור מעגל בין הסוללה לקבל כך שאלקטרונים ייזרמו מהסוללה לקבל מלמעלה ומהקבל לסוללה מלמטה והקבל יקבל את אותו המתח של הסוללה. אם ננתק את המתג הקבל ימשיך להחזיק את המתח, ואם נדליק את המתג השני יהיה לנו זרם קצר מהקבל אליו עם כיוון השעון (בחלקו העליון של הקבל היונים החיוביים ומתחתיו השליליים, והאלקטרונים הם אלו שנעים).



בהקשר של מחשבים נקבע מתח נמוך (עד $1.5v$) להיות 0 ומתח גבוה (לפחות $3.5v$) להיות 1 - "כי ככה". בין 1.5 ל-3.5 וולט לא אמורים להיות במצב יציב, רק במעבר בין המצבים. האנלוגיה כאן היא האם מיכל מים הוא מלא או ריק.

מוליכים למחצה

מוליך למחצה הוא חומר שלא מוליך חשמל מאוד טוב (מולכותו היא בין נחושת למוליך).

דוגמה סיליקון טהור הוא מוליך למחצה כי בצורת הגביש עליה דיברנו יש מעט מאוד אלקטרונים חופשיים (הרבה מהם תפוסים בין כמה אטומים בקשר קוולנטי) ולכן קשה להזרים דרכו זרם.

אילוח (doping) הוא תהליך שבו ״מלכלכים״ את הסיליקון.

- P-type : נוסיף לסילקון קצת אלומיניום (לו 3 אלקטרונים מתוך 4 במסלול האחרון) כך שייקשר לאטומי הסיליקון כך שלחומר עכשיו יהיה חסר אלקטרון והוא ישמח לקבל אותו, ואז אלקטרונים יעברו בקלות בין האי-שלמויות האלה (פעם ישלים את המחסור במוקד לכלוך אחד, ואז יקפוץ לאחר, וכו׳). החורים שנוצרים כשאין את האלקטרון הנדרש ליציבות נעים (לשם התאוריה), בכיוון ההפוך מהאלקטרונים וכך נוצר זרם חיובי בכיוון ההפוך מתנועת האלקטרונים.
- N-type : העיקרון הנ״ל רק עם זרחן (לו 5 אלקטרונים כלומר 1 במסלול האחרון) כך שיווצר עודף אלקטרונים והאלקטרונים העודפים חופשיים לנוע לאן שירצו ויצרו זרם.

למוליך מחצה מסוג P ו-N אין מטען חיובי או שלילי אלא רק סיבות שונות לתנועת האלקטרונים כתוצאה מהרצון להשלים מסלולים אחרונים באטומים.

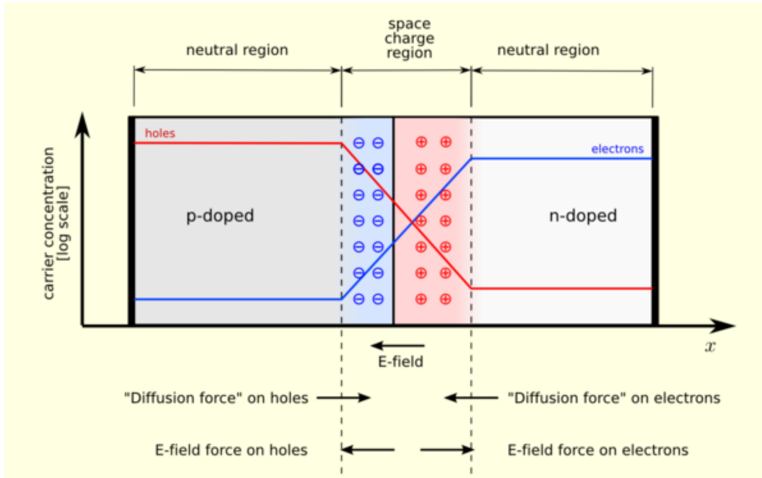
הצמדה של חומרים מסוג P ו-N יוצרת יונים - האלקטרונים מ-N שמחים לקפוץ ל-P שם ״צריך״ אותם. שני כוחות פועלים בעת הצמדת חומרים שכזו :

- הרצון של המסלולים להתמלא - מה שגורם לאלקטרונים לקפוץ מ-N ל-P.
- הכוח החשמלי שיוצרים האלקטרונים - כוח דחייה בין מטענים שליליים שמונע קפיצת אלקטרונים מ-P ל-N (N אומר ״יש לי מספיק שליליים כבר״).

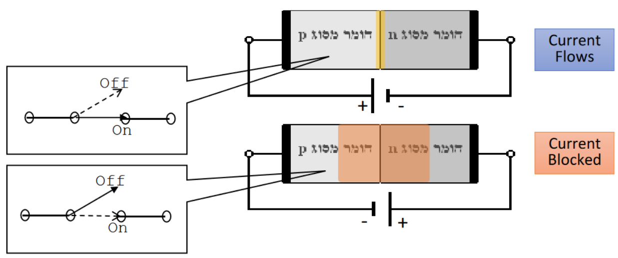
צומת p-n

ההצמדה הזו נקרא צומת PN (PN junction) והיא מוליכה מכיוון אחד וחוסמת זרם מכיוון אחר (כמו שסתום!). המנגנון שהצומת מספק נקרא דיודה (diode) ומסומן באיורים הבאים 

בשל תזוזה מקרית של אלקטרונים ופרוטונים בסמוך לצומת, מצליחים לעבור אטומים יוניים שליליים מ-N ל-P וחיוביים להפך (בניגוד לכיוון הזרימה). כשמספיק התחלפויות כאלה קורות, ישנה מאסה של אטומים יוניים חיוביים ב-N ומאסה של שליליים ב-P שלא יעברו לצד השני בגלל שהם חלק מהגביש כבר. בצורה זו נוצר מחסום מכוח כוח הדחייה החשמלי שגורם להפסקת הזרימה דרך הצומת והחזקת מתח בו (ראו איור)



עתה חיבור סוללה לדיודה נותן לנו תכונות מעניינות.



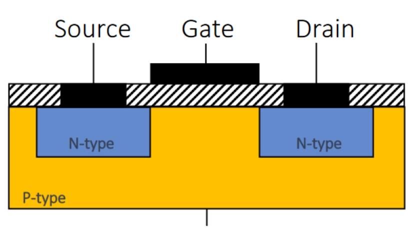
- חיבור סוללה עם + ל-P ו-– ל-N יגרום להרבה מאוד יונים חיוביים לזרום מ-P ל-N ויוניים שליליים מ-N ל-P (בהתאם לכיוון הזרימה לפני שנחסמה) וכך יצטמצם המרווח באמצע עד ל-0 ותהיה לנו זרימה רגילה, כאילו סגרנו מתג.
- חיבור סוללה עם + ל-N ו-– ל-P יגרום ליוניים חיוביים ושליליים להגיע לחומרים ולחזק את היונים במרווח שכרגע מונעים מעבר ורק יחזקו את החסימה, כך שלמעשה דימינו התנהגות של פתיחת מתג.

MOS-FET

טרנזיסטור MOS-FET עשוי משלושה חומרים: מוליך למחצה; תחמוצת מבודדת; ומתכת. יש לו בנוסף שלוש רגליים: source; drain; ו-gate (ורגל הארקה שמחוברת תמיד).

דרך הפעולה של ווריאנט ה-N-Channel

באיור ניתן לראות NMOS, ווריאנט מבין שניים של MOS-FET (השני משלים לו רק ש-N ו-P במיקומים הפוכים). החומר המקווקו הוא המבודד והשחור הוא המתכת.

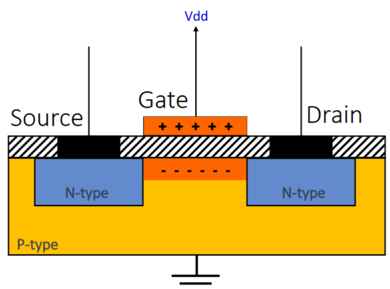


נזכור כי זרימה ניתן לקבל רק מ-P ל-N ולכן אי אפשר אף פעם להזרים שום דבר מהמקור (source) לשפך (drain) ולהפך. יש לנו למעשה שתי דיודות גב אל גב (שקצוותיהן המקור והשפך).

אם המתכת של השער מקבלת מטען, האזור בין השער למוליך-למחצה נהיה קבל כי הוא מחזיק הפרשי מטענים!

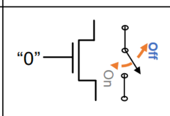
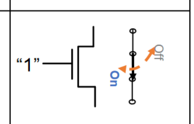
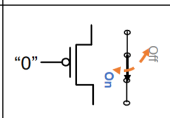
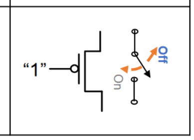
הערה את כל ארבעת הרגליים תמיד נחבר לאנשהו - או לאדמה או לסוללה או לטרנזיסטור אחר.

- אם נחבר את השער לאדמה (הארקה), יהיו לנו שני צמתי p-n שלא ניתן יהיה להזרים דרכן (ובפרט בין S ל-D דבר).
- אם נחבר את השער למתח, הקבל שהזכרנו למעלה מקבל מתח ועכשיו יש מטען חיובי מעליו ושלילי מתחתיו, כלומר ישנם אלקטרונים חופשיים ב-p-type, וכשאלו נוגעים בחומר n-type משני הצדדים יוצרים ערוץ אלקטרונים חופשיים דרכו כן יכול לעבור זרם, ומכאן השם n-channel.



מה שקיבלנו בסופו של דבר הוא סוויץ' שאנחנו יכולים לשלוט בו באמצעות זרם לשער (0 או 1). את הטרנזיסטור נסמן בסימול הבא - 

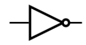
P-channel עובד אותו הדבר רק הפוך - הזרמת "0" לשער תסגור את המתג ו-"1" תפתח אותו. ההבדל המהותי היחיד חוץ מהחומרים הוא שיש מתח שמחובר מלמטה במקום הארקה. להלן טבלת סיכום של דרך הפעולה של הטרנזיסטורים.

Gate	"0"	"1" Vdd
N-MOS		
P-MOS		

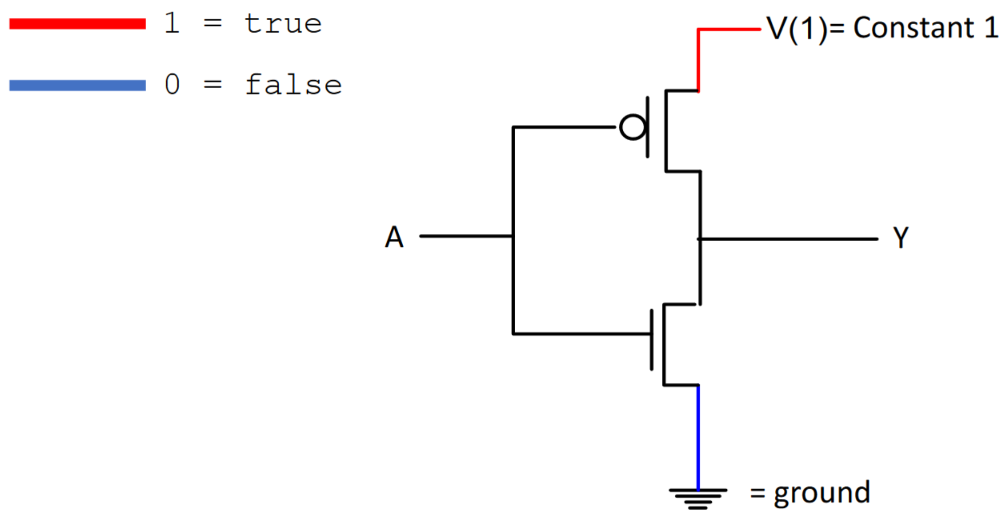
הערה בתחום ככלל, ○ מסמן שלילה, ב-PMOS למשל "שוללים" את המתח ומתנהגים הפוך ממנו. גם שערי NOT מסומנים עם עיגול.

בניית שערים מטרנזיסטורים

טרנזיסטורים הם שימושיים לנו כי אנחנו יכולים לבנות מהם שערים לוגיים.

דוגמה נבנה שער NOT (Inverter) שמסומן ב- 

הבנייה דורשת שני MOF-SET-ים (P-Channel למעלה ו-N-Channel למטה), והיא כבאיור.



A הוא הקלט ו-Y הוא הפלט. כש-A הוא ”1“, יעבור זרם של 0 (שהוא זניח) דרך ה-N-Channel למטה (השער דלוק) ולא יזרום שום דבר דרך ה-P-Channel למעלה (השער דלוק לכן אין זרימה). סה“כ אין שום זרימה עם מתח לא זניח לכן Y יהיה ”0“.

בהתאם אם A הוא ”0“ אז ה-PMOS יזרים דרכו זרם קבוע של ”1“ ו-Y יהיה ”1“.

הערה צריך את ה-NMOS התחתון כי אחרת ב-$A =$ ”1“ יש לנו מעגל פתוח שיכול להיות לו כל זרם ולא בהכרח ”0“ כמו שאנחנו רוצים.

תרגול

בסיס ספירה הוא דרך לייצג מספרים ממשיים. בבסיס עשרוני נבצע את ההמרה הבאה

$$(a_4a_3a_2a_1a_0.a_{-1}a_{-2}a_{-3})_{10} = a_4 10^4 + \ldots a_0 10^0 + a_{-1}10^{-1} + a_{-2}10^{-2} + a_{-3}10^{-3}$$

בסיס בינארי משתמש בסיביות (ביטים) שערכן 0 או 1, בעשרוני 0 עד 9, ובהקסדצימאלי $0-9, A-F$.

טווח המספרים בעל n ספרות בבסיס r הוא $\{0, \ldots, r^n - 1\}$

במקרה הכללי $a_m \ldots a_0.a_{-1} \ldots a_{-n}$ בבסיס r מייצג את המספר $\sum_{i=-n}^{m} a_i r^i$.

פעולות חשבוניות אפשר לעשות באותו האופן כבבסיס עשרוני (חיבור ארוך שבו עוברת ספרה הלאה, כאשר הספרות נגמרות לא בהכרח אחרי 9).

מעבר לבסיס 10 הוא די פשוט (פריסת הייצוג וחישוב מכפלות וחיבור בבסיס עשרוני). מעבר מבסיס 10 לבסיס אחר הוא פשוט תהליך איטרטיבי של פעולות מודולו r (הבסיס) כאשר מה שהוא לא השארית יהיו הספרה הראשונה (משמאל), השנייה, וכו' עד שנגמרות הספרות.

לא כללתי כאן אינסוף דוגמאות להמרות בין בסיסים כי לא מספיק משעמם לי.

במקרה הכללי נמיר משני בסיסים כלשהם דרך בסיס 10 כאשר את הרכיב משמאל ומימן לספרה העשרונית נטפל באופן נפרד וזהה (עד כדי חלוקה איטרטיבית בשמאל ומכפלה איטרטיבית בימין).

דוגמה נמיר את $(0.79272)_{10}$ לבסיס 4. נכפיל את המספר ב-4 ונקבל 3.17088. לכן הספרה הראשונה היא 3 והשארית היא 0.17088. נבצע זאת שוב ועתה נקבל 0 ושארית 0.68352, וחוזר חלילה עד שהשארית תהיה 0, כאשר עתה הספרות הן מלמעלה למטה במקום למטה למעלה בספרות הרגילות.

שיטות לייצוג מספרים

- שיטת גודל וסימן: מספר יתחיל בביט סימן (0 חיובי 1 שלילי) ושאר הביטים יהיו ייצוג בינארי של המספר.

  דוגמה $9 = (-1)^0 \left(2^3 + 2^0\right) = 01001$.

  טווח הייצוג בשיטה זו הוא $\left(2^{n-1} - 1\right), \ldots, 0, \ldots, \left(2^{n-1} - 1\right) -$.

- שיטת המשלים לאחד: ביט סימן, שלפי ערכו נדע האם שאר הספרות הן הערך הבינארי של המספר וזהו, או שזהו הערך הבינארי של המשלים של המספר ל-$\left(2^{n-1} - 1\right) -$, ובנוסף הפיכת כל ביט תספק לנו את השלילה של המספר. לדוגמה, $4 = 00100$, ואם נהפוך כל ביט נקבל $-4 = 11011$.

  חיסור מספרים הוא די פשוט: צריך לחבר את המספרים, ואם יש carry לאחר החישוב נמחק אותו ונוסיף אחד לתוצאה.

  דוגמה $5 = (-4) + (+9) = 9 - 4$ כי $01001 + 11011 = 100100$ וזה הופך ל-00101.

  טווח הייצוג הוא כמו בשיטת גודל וסימן ויש בו, כמו בשיטה הקודמת 0 חיובי ו-0 שלילי.

- שיטת המשלים לשתיים: ביט סימן, שעבור מספרים שליליים ערכו הנגדי למספר שמתקבל מהיפוך הביטים והוספת אחד.

  דוגמה $-4 = -00100 = -(00011 + 1) = 11100$.

  לחלופין לחישוב המשלים אפשר ללכת מימין לשמאל עד ל-1 הראשון, לא לשנות אותו, ואז להפוך את כל מה שמשמאלו (ואז לא צריך להוסיף 1).

  דוגמה למה שווה -5 בשיטת המשלים ל-2? $5 = 00101$, נוסיף אחד ונשנה את ביט הסימן ונקבל 10110.

  כדי לחסר מספרים, נסכום אותם ונמחק carry אם יש.

הגדרה overflow הוא מצב שבו אנחנו סוכמים שני מספרים באותו הסימן ומקבלים מספר בסימן הפוך, במקרה זה התוצאה כמובן שגויה.

דוגמה נשתמש בשיטת המשלים ל-1 עם 5 ביטים, $01011 + 01101 = 11000$ כלומר סכמנו חיוביים וקיבלנו תוצאה שלילית!

הערה הפתרון ל-overflow הוא הוספת ביטים כך שיגדל טווח הייצוג.

שבוע III | שערים

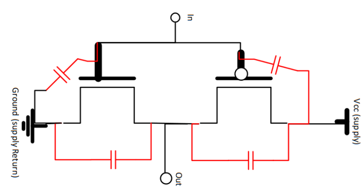
הרצאה

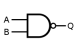
הערה כל שער שנבנה נבנה סימטרית מבחינת השימוש ב-PMOS ו-NMOS, לכן השערים עם טרנזיסטורים אלה נקראים Complementary MOS.

הערה לעולם לא נרצה להזרים זרם אל תוך טרנזיסטור אחר דלוק (מהכיוון הלא נכון) כי זה גורם לקצר.

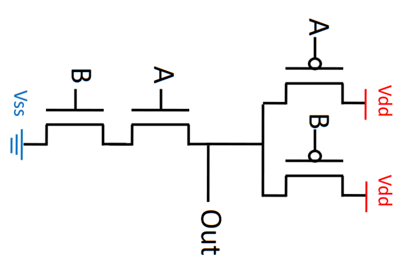
שער אחד בפני עצמו זה נחמד, אבל מעבדים בונים ע"י חיבור שערים. את השערים נחבר עם חומר מוליך בין הקלט של שער אחד לפלט של השער שממנו אנחנו לוקחים את התוצאה.

בין שער למקור, בין מקור לשפך ובין שער לשפך נוצרים קבלים אפקטיביים (לא הוספנו שום דבר, פשוט יש רכיבים פיזיים שמוצאים עצמם מחזיקים מתח בין הצדדים). את הקבלים האלה לוקח זמן לטעון או לפרק ממתח בעת שינוי מצב (שינוי הזרם מהשער, הזרם מהמקור). לכן במצבים רגעיים יש זרימה שאנחנו לא בהכרח מצפים לה במהלך המעבר (10^{-15} של שנייה). ראו באיור באדום את הקבלים שנוצרים.

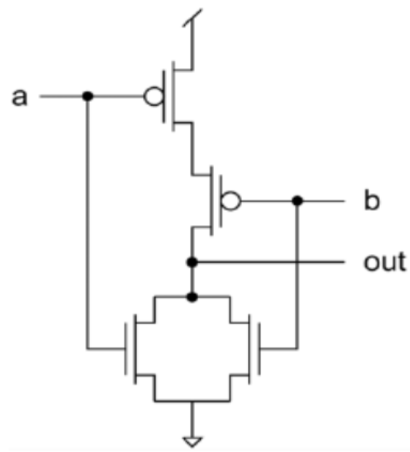


השער היסודי שמאפשר לבנות את כל שער השערים הוא NAND (Not And) - שנותן 0 אם"ם שני הפלטים 1 (ואחרת 1), ומסומן כך 

ניתן לבנות את NAND עם CMOS באופן הבא (הריצו פלטים כדי לראות שזה עובד).



בדומה ניתן לבנות שער NOR באופן הבא (קו אלכסוני הוא V (1) כלומר מתח קבוע דלוק וחץ הוא GND (0) כלומר הארקה)



הגדרה פ' בולינאית היא פ' שמקבלת קלטים בולינאני (משתנה שיכול לקבל אחד מבין שני ערכים, ביטים לדוגמה) ופולטת פלט בוליאני אחד בדיוק.

הערה מעתה נסמן x' להיות ההפוך ל-x (כלומר שהפעלנו עליו שער NOT).

דוגמה $F = xy'$ היא פ' שבהינתן x, y, פולטת פלט שערכו x וגם לא y.

סכום מכפלות ומכפלת סכומים

הגדרה בהינתן משתנים בולינאים לפ' בולינאית כלשהי, minterm הוא מכפלה (AND) של ליטרלים, כאשר ליטרל הוא משתנה או היפוכו וכל משתנה מופיע בדיוק פעם אחת כליטרל או בתוך ליטרל מהופך.

דוגמה עבור שלושה משתנים וההשמה $x = 1, y = 1, z = 0$, ה-minterm היחיד שערכו 1 יהיה $m_6 = xyz'$, ועבור $x = 1, y = 1, z = 1$ זה יהיה $m_7 = xyz$ (אינדקס ה-minterm עם ערך 1 מתקבל ע"י הערך הדצימלי של הסטרינג הבינארי המתקבל מהצמדת ההשמות במשתנים).

נשים לב ש-minterm מקבל ערך 1 בבדיוק שורה אחת של טבלת אמת מלאה על המשתנים.

הגדרה בהינתן השמה במשתנים, maxterm הוא סכום (OR) של המשתנים או היפוכהם כך שכל משתנה מופיע פעם אחת בדיוק.

דוגמה עבור שלושה משתנים עם ההשמה $x = 1, y = 1, z = 0$, ה-maxterm היחיד שמקבל ערך 0 הוא $M_6 = x' + y' + z$ וכו'.

הערה m_i משלים ל-M_i.

הגדרה Sum of Products הוא סכום מכפלות minterm-ים ו-Product of Sums הוא מכפלת maxterm-ים.

דוגמה הפ' F_1 עם טבלת האמת הבאה

x	y	z	F_1	Minterm	Maxterm
0	0	0	0	$m_0=x'y'z'$	$M_0=x+y+z$
0	0	1	1	$m_1=x'y'z$	$M_1=x+y+z'$
0	1	0	0	$m_2=x'yz'$	$M_2=x+y'+z$
0	1	1	1	$m_3=x'yz$	$M_3=x+y'+z'$
1	0	0	1	$m_4=xy'z'$	$M_4=x'+y+z$
1	0	1	0	$m_5=xy'z$	$M_5=x'+y+z'$
1	1	0	1	$m_6=xyz'$	$M_6=x'+y'+z$
1	1	1	0	$m_7=xyz$	$M_7=x'+y'+z'$

ניתנת לייצור ע"י

$$F_1(x,y,z) = m_1 + m_3 + m_4 + m_6 = x'y'z + x'yz + xy'z' + xyz'$$

הטכניקה הייתה לסכום את כל ה-minterm-ים שמקבלים 1 בפ' (בכחול).

לחלופין נוכל לייצג את הפ' עם PoS ע"י הכפלת כל ה-maxterm-ים שמקבלים ערך 0 (באדום) כלומר

$$F_1(x,y,z) = M_0 \cdot M_2 \cdot M_5 \cdot M_7$$

כלומר הצגנו בצורה קנונית פ' לכאורה מורכבת!

הערה למה משמשים כל הייצוגים השונים (טבלת אמת, PoS, SoP ומפת קרנו שנלמד בתרגול)? נרצה בסופו של דבר את הייצוג המינימלי, כך שנדרש למספר הקטן ביותר של שערים כדי לממש אותו.

דוגמה מולטיפלסקר (MUX) מקבל שלושה קלטים: $S(elect), D0, D1$ שפלטו $D0$ אם $S=0$ ואחרת $D1$. בטבלה X משמעו "לא משנה מה הערך"

S	D1	D0	Y
0	X	0	0
0	X	1	1
1	0	X	0
1	1	X	1

וניתן לרשום את הפ' כ-$\text{MUX}(S, D_0, D_1) = S \cdot D_1 + S'D_0$, ואת זה ניתן לממש באמצעות שערים שכבר בנינו. עם זאת, ניתן לממש את המעגל עם פחות טרנזיסטורים מאשר במימוש נאיבי עם שני AND-ים, OR ו-NOT.

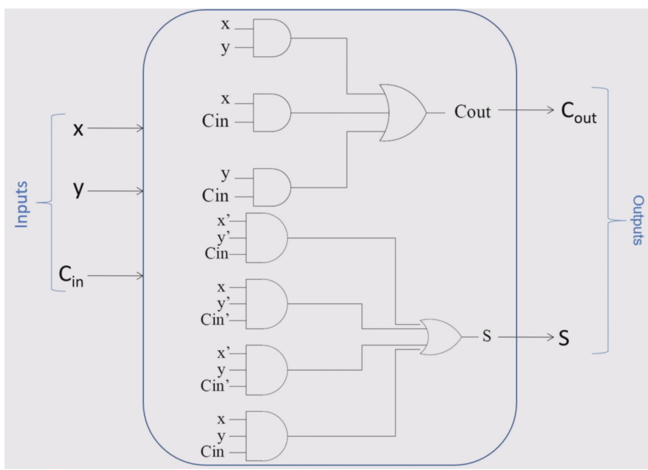
דוגמה Full Adder הוא מעגל שמקבל x, y, C_{in} (כאשר C_{in} נשא מסכימה קודמת) ופולט S, C_{out} כאשר S הוא הסכום ו-C_{out} הוא הנשא (overflow) מתוך הסכום (ראו טבלת אמת)

Truth Table

x	y	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

ניקח את $x=0, y=1, C_{in}=1$, אז $x+y+C_{in}=0+1+0=(10)_2$ ולכן $S=0, C_{out}=1$.

למעשה, בגלל שיש למעגל שני פלטים, הרי שהוא מורכב משתי פ' בוליאניות. כרגיל אפשר לממש את המעגל באמצעות SoP (סכימת minterm-ים), ובמימוש הבא צמצמנו כמה minterm-ים באמצעות מפות קרנו שנלמד בהמשך (בנוסף מופיע במעגל OR עם שלושה וארבע כניסות - הסטודנטית המשקיעה תראה כיצד ניתן לעשות זאת עם פי 2 טרנזיסטורים ממספר הכניסות).



הערה קל לשרשר כמה FA-ים כדי לקבל מעגל שסוכם מספרים עם מספר ביטים גדול יותר (לוקחים את C_{out} של ה-FA על זוג הביטים הראשונים, מכניסים אותו ל-FA וחוזר חלילה).

יחידות סטנדרטיות במעגלים לוגים

1. מפענח (Decoder) : הקלט הוא n ביטים $x_0, \ldots, x_{n-1}$ והפלט הוא $d_0, \ldots, d_{k-1}$ כאשר $k=2^n$ ו

$$d_j = \begin{cases} 1 & (j)_{10} = (x_{n-1} \ldots x_0)_2 \\ 0 & \text{אחרת} \end{cases}$$

כלומר פורש וקטור של n ביטים על 2^n ביטים שכל אחד מייצג מספר מתוך $\{0, \ldots, 2^n - 1\}$.

ברגע שיש לנו בלוק של מפענח, אפשר להשתמש בו כדי לממש פ' בולינאית באופן טריוויאלי, כי כל מה שצריך לעשות זה לעשות OR על כל ה-d_i שמייצגים $x_0, \ldots, x_{n-1}$ שמקבל ערך 1 בטבלת האמת של הפ' הבוליאנית.

2. מפלג (DeMultiplexer): הקלט הוא $x, s_0, \ldots, s_{n-1}$ כל אחד בגודל ביט והפלט הוא $f_0, \ldots, f_{k-1}$ כאשר $k = 2^n$ ו

$$f_j = \begin{cases} x & (j)_{10} = (s_{n-1} \ldots s_0)_2 \\ 0 & \text{אחרת} \end{cases}$$

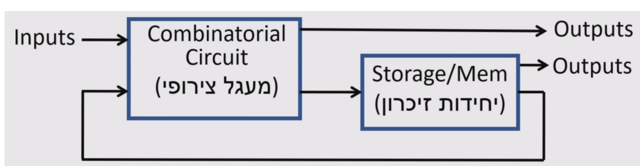
כלומר בהינתן ביט, מחזיר מחרוזת שבה הכל אפסים חוץ מאולי הביט ה-$(s_{n-1} \ldots s_0)_2$ שערכו x.

3. מקודד (Encoder): הקלט הוא ביטים $x_0, \ldots, x_{k-1}$ כאשר $k = 2^n$ והפלט הוא ביטים $e_0, \ldots e_n$ כאשר אם $x_i = 1$ עבור i אחד בדיוק (וכל השאר אפסים) אז $(e_{n-1} \ldots e_0)_2 = (i)_{10}$, כלומר מחזיר את האינדקס (בבינארי) של הביט היחיד הדלוק בקלט.

4. מרבב (Multiplexer): הקלט הוא $k = 2^n$ ביטים $x_0, \ldots, x_{k-1}$ וביטים $s_0, \ldots, s_{n-1}$ והפלט הוא f שהוא ערך הביט עם אינדקס $(s_{n-1} \ldots s_0)_2$ ב-x.

מעגלים סדרתיים

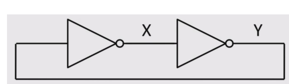
הגדרה מעגל סדרתי הוא מעגל קומבינטורי שחלק מהקלטים שלו הם פלטים של יחידת זיכרון שמחזיק השער (ראו איור)



מעגלים סדרתיים אסינכרונים יכולים לשנות מצב בכל זמן, ואילו מעגלים סינכרוניים משנים מצב בהתאם לשעון.ד

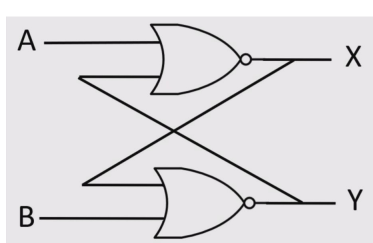
הגדרה שעון סיגנל בצורת גל מחזורי (0 ואז 1 ואז 0 ואז 1 וכו').

דוגמה כיצד השער הבא יתנהג?



אם הקלט בהתחלה הוא $X = 0$ אז $Y = 1$ ואז $X = 0$ ונקבל מצב יציב של $(X, Y) = (0, 1)$. בדומה עבור $X = 0, Y = 1$.
כלומר, יש לנו מערכת דו-יציבה (עם שני מצבים יציבים), שיכולה לשמש אותנו לאכסון זכרון.

דוגמה נביט בשער הבא, שנקרא SR Latch (נזכיר שהשערים במעגל הם NOR-ים)

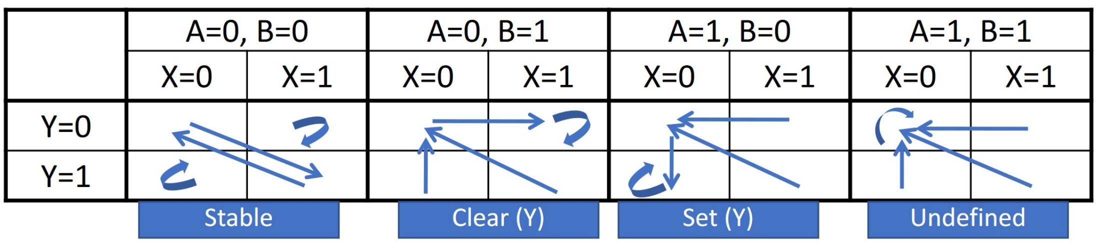


הפ׳ הזו מקיימת $Y(t+1) = (B+X(t))'$, $X(t+1) = (A+Y(t))'$ (כאשר $X(t)$ הוא ערכו של X לאחר שינוי קלטים מסונכרן עם שעון שביצע t טיקים), או בטבלת אמת עמוסה

A	B	X(t)	Y(t)	X(t+1)	Y(t+1)
0	0	0	0	1	1
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	1	0
0	1	0	1	0	0
0	1	1	0	1	0
0	1	1	1	0	0

A	B	X(t)	Y(t)	X(t+1)	Y(t+1)
1	0	0	0	0	1
1	0	0	1	0	1
1	0	1	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	0	0

ואם נשלוף רק את הערכים המעניינים, נוכל להביט בתופעה מעניינת (חצים מעגליים משמע לאחר טיק נשאר באותם הערכים וחצים אומרים שנעבור לזוג ערכים אחר)



ונשים לב שאם $X=Y$ נקבל מצב לא יציב (לא משנה מה ערכי A,B תמיד יש חץ שיזיז אותנו למצב אחר) ולכן תמיד נניח שאנחנו משתמשים ב-SR Latch כאשר $Y=X'$ (זה דומה למצב בדוגמה הקודמת שבו $X=Y=0$ שזה לא מוגדר בכלל כי יש לנו מהפך עם אותו הערך משני הצדדים).

אם כן, עבור $(A,B)=(0,0)$, נקבל ש-(X,Y) שומר על ערכו (כזכור אנחנו מתעלמים מ-$X=Y$), ואם $(A,B)=(0,1)$ אז אנחנו מאפסים את Y , ואם $(A,B)=(1,0)$ אז מדליקים את Y ואם $(A,B)=(1,1)$ נקבל מצב לא מוגדר שנתעלם ממנו.

לכן, באמצעות $(S,R)=(A,B)$ נוכל לשלוט בערכו של Y כשיש לנו זכרון של המצב הקודם, עם טבלת אמת מצומצמת נוחה ביותר, כאשר $Q(t)=Y(t)=X(t)'$ (בהתעלם מהמקרים הלא חוקיים)

S	R	Q(t+1)	Q’(t+1)
0	0	Q(t)	Q’(t)
0	1	0	1
1	0	1	0
1	1	0	0

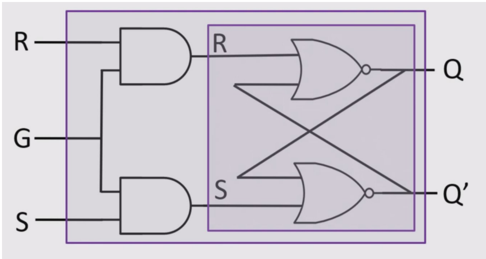
הערה מעגל סדרתיים ניתן לייצג באמצעות טבלת עירור (הטבלה הנ״ל) וטבלת מעברים, שעונה על השאלה ”אילו קלטים נדרשים כדי לעבור בין מצב כלשהו לאחר“ (Φ הוא ערך Don’t care)

From Q(t)	To Q(t+1)	S	R
0	0	0	Φ
0	1	1	0
1	0	0	1
1	1	Φ	0

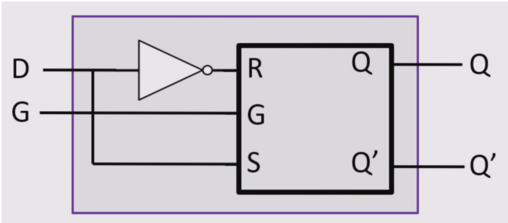
הערה יש היגיון בשמות הביטים S,R - אם רק S(et) דלוק, קובעים את הפלט להיות 1, אם רק R(eset) דלוק קובעים את הפלט להיות 0, ואם לא זה ולא זה דלוקים לא עושים כלום, כלומר שומרים על המצב כמות שהיה. כמובן שתחת סימולים אלה, גם S וגם R דלוקים זה לא מוגדר היטב.

מעגלים סדרתיים מורכבים על בסיס SR Latch

- **Gated SR Latch** הוא שער שמונע פליטת ערכים לא חוקיים מ-SR Latch, והוא מקבל קלטים S,R ו-G כאשר G הוא ביט שער שאם הוא דלוק אז המעגל מתפקד כ-SR Latch רגיל, ואם כבוי אז הפלטים לא משתנים לא משנה מה. המימוש הוא די פשוט, וכולל AND של S,R עם G לפני הכניסה למעגל הפנימי (ראו איור)

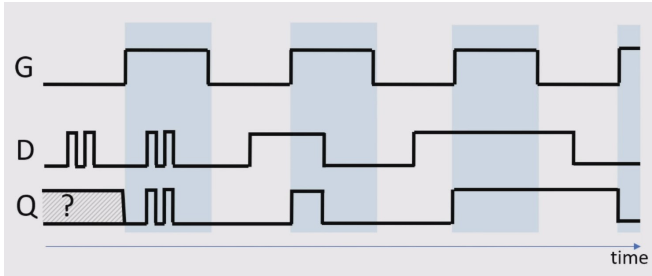


- **Gated D-Latch** הוא גרסה אפילו יותר בטוחה ל-Gated SR Latch - במקום לקבל קלטי S,R, נקבל D שיהיה מחובר ל-S ודרך מהפך ל-R, וכך לא נקבל מצב של $(1,1)$, וכדי לקבל $(0,0)$ אפשר פשוט לכבות את G (ראו איור)



הערה לרוב נחבר את השעון ל-G, כלומר אפשר לשנות את הערך רק כשהשעון בטיק שערכו 1.

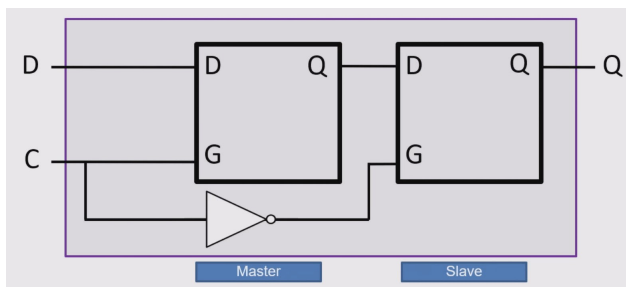
דוגמה בהינתן שעון שמחובר ל-G וביט D שערכו משתנה, נוכל לחשב מה יהיה הפלט Q, כפ' של הזמן. השטח המקווקו בהתחלה פירושו שלא ידוע לנו מה הערך של Q שכן הוא לא מוגדר בשלב הזה.



D Flip Flop

נבנה מעגל Shifter, שהוא מעגל שבו ביט הקלט זז צעד אחד ימינה בכל מחזור. אם נשרשר D Latch-ים שכל ה-G-ים שלהם מחוברים לשעון, הקלט יופיע מיד בסוף השרשור (למעט זמן פעפוע של החשמל שהוא זניח).

הפתרון לזה הוא להוסיף מהפך בין השעון ל-Latch השני כך שכשהשעון ב-1 רק ה-Latch הראשון יוכל לשנות את ערכו וכשהשעון ב-0 רק ה-Latch השני ישנה את ערכו והראשון לא ישתנה. שער כזה נקרא D Flip Flop.



לכן רק בעת ירידת השעון נקבל שינוי של הפלט Q, כי לאחר העליה ה-Master ישנה את הפלט ולאחר הירידה ה-Slave ישנה את הערך, הלא הוא פלט המעגל כולו. כל עוד השעון לא ירד, הערך ישאר זהה לערך שקיבל בירידת השעון האחרונה.

הערה ניתן לבנות מעגל שישתנה רק בעליה באמצעות הזזת המהפך ללפני השער של המאסטר ולא העבד.

תרגול

הגדרה אלגברה בוליאנית היא מבנה אלגברי המוגדר על קבוצת איברים B עם שני אופרטורים בינאריים $\cdot, +$ כשלכל $x, y, z \in B$ מתקיימות אקסיומות Huntington: סגירות לכפל וחיבור, קיום איברי יחידה לכפל וחיבור, קומטטיביות בחיבור וכפל, דיסטריוטיביות (שני הנוסחים), קיום משלים (כך ש-$x \cdot x' = 0, x + x' = 1$) ו-$|B| \geq 2$.

הגדרה אלגברה בוליאנית דו-ערכית מוגדרת על קבוצה בת שני איברים $B = \{0, 1\}$ עם האופרטורים $x \cdot y = \text{AND}(x, y), x + y = \text{OR}(x, y)$ ו-$x' = \text{Not}(x)$.

טענה באלגברה בולינאית דו ערכית מתקיימות התכונות הבאות:

- אידמפוטנטיות: $x \cdot x = x + x = x$ לכל x.

- $x\cdot 0=0, x+1=1$.
- אסוציאטיביות לחיבור וכפל.
- חוק הצמצום: $x(x+y)=x, x+xy=x$.
- $(x')'=x$.

הערה סדר האופטורים בחישוב ביטוי בוליאני הוא קודם סוגריים, אז NOT, אז AND ואז OR.

דרכים לבטא פונקציה בוליאנית

- ביטוי בוליאני (ביטוי על המשתנים עם שני האופרטורים ושלילה).
- טבלת אמת: לטבלה יהיו 2^n שורות כאשר יש n קלטים.
- סכום מכפלות: מספיק שמכפלה אחת תהיה 1 כדי שהסכום יהיה 1. כדי להגיע לייצוג סכום מכפלות, סוכמים את כל המכפלות הסטנדרטיות (minterm) שמקבלות 1 בטבלת האמת.

  המשתנה ה-j מקבל שלילה ב-minterm ה-i אם"ם הביט ה-j ב-$(i)_2$ הוא 0.
- מכפלת סכומים: מספיק שסכום אחד יהיה 0 ואז כל המכפלה היא 0. כדי להגיע לייצוג, מכפילים את כל הסכומים הסטנדרטיים שמקבלים 0 בטבלת האמת. ניתן להגיע לשקילות לסכום מכפלות עם שלילה על הביטוי ושימוש בשני כללי דה-מורגן.

  המשתנה ה-j מקבל שלילה ב-maxterm ה-i אם"ם הביט ה-j ב-$(i)_2$ הוא 1.

נרצה לצמצם את מספר הליטרלים שלנו (מספר השערים).

דוגמה נביט ב-$F1(x,y,z)=x'y'z+x'yz+xy'$ ו-$F2(x,y,z)=xy'+x'z$. את הפ' השניה ניתן לממש עם משמעותית פחות שערים לוגיים (יש הרבה פחות פעולות) אבל הפ' שקולות ולכן נעדיף את השניה תמיד!

את השקילות אפשר להראות עם טבלת אמת או באמצעות אלגברה בוליאנית:

$$\begin{aligned}
xy'z+x'yz+x' &= x'zy'+x'zy+xy' \\
&= x'z\underbrace{(y'+y)}_{1}+xy' \\
&= x'z+xy' \\
&= F2(x,y,z)
\end{aligned}$$

דוגמה נפשט עוד פ'

$$\begin{aligned} F\left(x,y,z\right) &= \left(x+y\right)\left[x'\left(y'+z'\right)\right]'+x'y'+x'z' \\ \text{דה מורגן} &= \left(x+y\right)\left[x+\left(y'+z'\right)'\right]+x'y'+x'z' \\ \text{דה מורגן} &= \left(x+y\right)\left(x+yz\right)+x'y'+x'z' \\ \text{דיסריבוטיביות} &= \left(xx+xyz+yx+yyz\right)+x'y'+x'z' \\ &= \ldots = 1 \end{aligned}$$

מפות קרנו

מפת קרנו בונים פעם אחת לכל הפ' הבוליאניות ב-n משתנים. ל-n משתנים יש מפת קרנו עם 2^n משבצות.

ראשית נבנה טבלת אמת לפ' הבוליאנית, ואז נציב את ה-minterm-ים בשבלונה של מפת הקרנו אם נצליח לשנן אותה.

				y	
		0	0	1	1
	0	m_0	m_1	m_3	m_2
x	1	m_4	m_5	m_7	m_6
		0	1	1	0
			z		

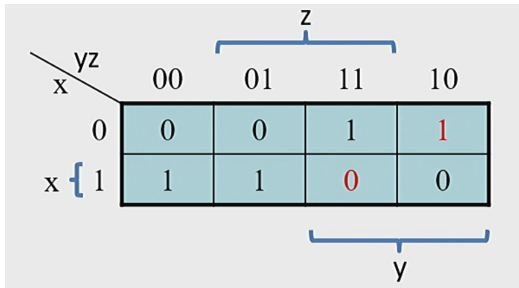
לחלופין נוכל לבנות מחדש את הטבלה עם האינטואיציה שבסימנים הכחולים למשתנים, באמצעות ערכי x ו-yz ניתן להסיק בקלות את ה-minterm-ים (יש לשים לב שערכי yz הם לא לפי סדר לקסיקוגרפי)

		z		
x \ yz	00	01	11	10
0	x'y'z'	x'y'z	x'yz	x'yz'
x 1	xy'z'	xy'z	xyz	xyz'
			y	

הערה ארבעת המשבצות ש-z מסומן עליהן נסכום לליטרל יחיד שהוא z, כך גם על y, וכך גם השורה התחתונה נסכמת ל-x. נשים לב גם כי כל שני ריבועים סמוכים נבדלים בליטרל אחד בלבד.

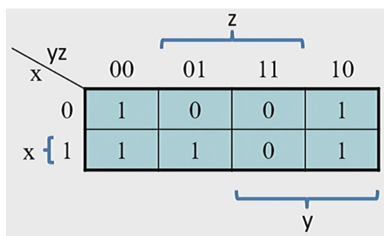
כדי לבצע צמצום נמצא קבוצות של ריבועים סמוכים שערכם בטבלת האמת 1 עבור הפ', כשגודל הקבוצה חייב להיות חזקה של 2 (כולל 1) ועלינו לבחור קבוצות גדולות ככל האפשר. קבוצות יכולות לחפוף וצריך לכסות את כל הריבועים. הטבלה היא מעגלית ולכן מלבן יכול לחצות את הקצה מימין ולהמשיך משמאל.

דוגמה עבור $F(x,y,z)=\sum(2,3,4,5)$ (סכום מכפלות עם אינדקסים). מפת הקרנו שלנו היא הבאה, כאשר הגענו אליה או באמצעות השבלונה או באופן הבא: הסימונים של x,y ו-z אומרים לנו איפה המשתנה מקבל ערך 1, ולכן במשבצת השנייה מימין בשורה העליונה לדוגמה, גם y וגם z הם 1 אבל x הוא 0, כלומר מדובר במקרה של $(0,1,1)$ שבמקרה שלנו זה 1 (כי זהו ערכו של ה-minterm השלישי שמהגדרת הפ' הוא 1).



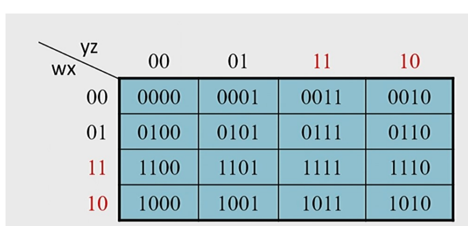
כדי לצמצם את הטבלה עכשיו נכסה את הטבלה עם שני מלבנים, אחד משמאל למטה ואחד מימין למעלה וכך נקבל ייצוג מינימלי של הפ' הבוליאנית שהוא $F(x,y,z)=x'y+xy'$ (במלבן משמאל משותף הכל חוץ מ-z וכך גם במלבן מימין).

דוגמה $F(x,y,z)=\sum(0,2,4,5,6)$. הביטוי הלא מצומצם מכיל 5 ביטויים. נמלא איכשהו את הטבלה ונקבל

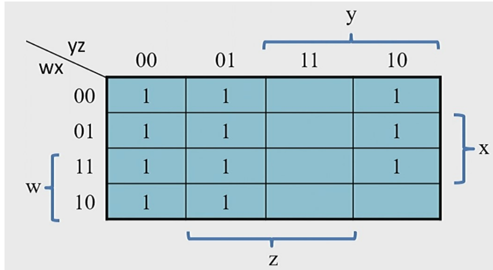


נשתמש בחפיפות וגם במעגליות ונקבל מלבן אחד משמאל למטה ועוד מלבן שכולל את העמודה השמאלית והעמודה הימנית יחד, ואז הביטוי המינימלי הוא $F(x,y,z)=xy'+z'$ (בשמאל למטה נופל z ובמעגלי נופלים x ו-y, פשוט עוברים על זוג ריבועים אופקי וזוג אנכי ורואים מה משתנה בהם, ומה משותף בהם).

מפת קרנו בארבעה משתנים נראת כך, כאשר אפשר לזכור אותה באמצעות העובדה שערכי הצירים שלה (גם אופקי וגם אנכי) הם $0,1,3,2$ (הייצוג הבינארי של המספרים).



דוגמה $F(x,y,z)=(0,1,2,4,5,6,8,9,12,13,14)$ מקבלת מפת קרנו עם הערכים הבאים (לא משנה איך מגיעים לזה)



נבחר את חצי המפה השמאלית כי זו שמיניה והמלבן הגדול ביותר שיש, ובשביל הערכים מימין נבחר שתי רביעיות מעגליות (אי אפשר את כולם ביחד כי זה נותן לא חזקה של 2).

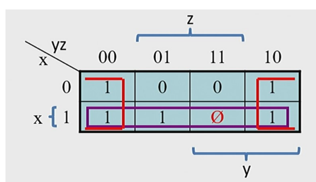
החצי השמאלי נבדל ב-z, w ו-x, ו-y מקבל 0 כלומר הביטוי הראשון הוא y'.

הרביעיה המעגלית העליונה נבדלת ב-x ו-y ו-z, w מקבלים ערכים 0 שניהם, כלומר יש לנו $z'w'$ והרביעיה המעגלית התחתונה שונה ב-w ו-y ו-x ו-z מקבלים ערכים 1 ו-0 בהתאמה, כלומר הביטוי הוא xz'.

סה"כ קיבלנו $F\left(x,y,z,w\right) = y' + z'w' + xz'$.

הגדרה לפעמים עבור צירופים מסוימים, לא יהיה אכפת לנו מה הוא פלט הפ', צירופים כאלה נקראים צירופים אדישים וניתן להשתמש בערך שיותר נוח לנו איתו כך שיבוטלו ליטרלים רבים ככל האפשר. נסמן צירוף כזה ב-$\varnothing$.

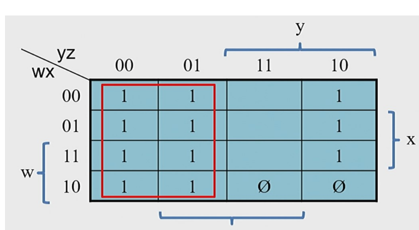
דוגמה $F\left(x,y,z\right) = \sum\left(0,2,4,5,6\right)$ היא הפ' הבוליאנית והצירופים האדישים הם $d\left(x,y,z\right) = \sum\left(7\right)$ (כלומר רק $\left(1,1,1\right)$).



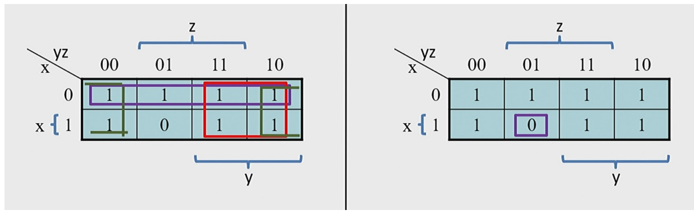
עתה נוכל לבחור מלבנים יותר נוחים (ראו איור) מאשר בהיעדר הצירוף האדיש כי אז לא היינו יכולים לבחור את השורה התחתונה כמלבן והיו לנו אמנם עדיין שני ביטויים אבל עם יותר ליטרלים, כלומר יותר שערים שזה פחות טוב.

הערה צירוף אדיש מתקבל לדוגמה כשברכיב אלקטרוני איזשהו מעגל בכל מקרה מחובר להארקה כך שלא משנה ערכו עבור צירוף מסוים כלשהו.

דוגמה נביט במפת הקרנו הבאה עם שני צירופים אדישים. הבחירה הכי נוחה היא 0 לשמאלי ו-1 לימני כי כל קומבינציה אחרת היתה דורשת מאיתנו יותר משני מלבנים או מלבנים קטנים יותר (יותר ליטרלים) *שזה פחות אידאלי.



דוגמה אפשר להשתמש במפת קרנו גם כדי לצמצם מכפלה של סכומים. נביט ב-$F(x,y,z) = \sum(0,1,2,3,4,6,7) = \Pi(5)$. כפי שניתן לראות, צמצום של סכום המכפלות דורש לפחות 3 נסכמים ואילו מכפלת סכומים דורש בדיוק ליטרל אחד.



כשמצמצמים פ' בוליאנית לפי מכפלת סכומים, מבצעים בדיוק את התהליך של סכום מכפלות רק שמססכים 0-ים במקום 1-ים. לאחר הכיסוי, ממירים את הכיסוי למכפלה של סכומים, כאשר כל סכום מתאים למלבן אחד.

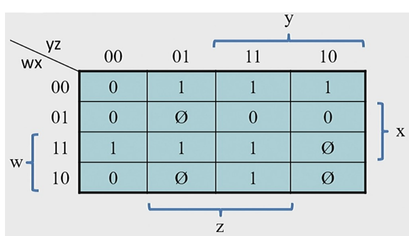
הערה ניתן להוכיח נכונות של צמצום לפי מפת קרנו למכפלת סכומים או באמצעות דה-מורגן לצמצום של סכום מכפלות, או פשוט באופן ישיר מטבלת האמת ונכונות ייצוג ה-maxterm-ים.

דוגמה נתונה הפ' הבוליאנית

$$F(w,x,y,z) = \sum(1,2,3,11,12,13,15) + d(w,x,y,z)\,, \quad d(w,x,y,z) = \sum(5,9,10,14)$$

- ציירו את מפת קרנו עבור הפ' F.

המפה תראה כך

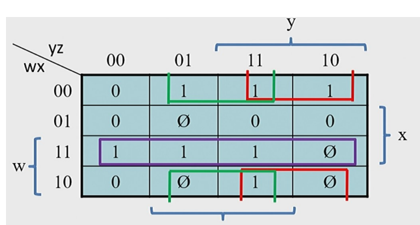


- כמה פ' שונות מיוצגות ע"י המפה?

$16 = 2^4$ כי אפשר לבחור ערכים שרירותיים עבור כל אחד מהצירופים האדישים שהוא ב"ת באחרים.

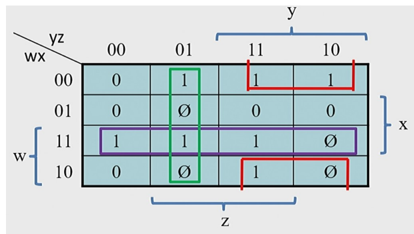
- רשמו סכום מכפלות מינימאי עבור F, האם הסכום יחיד?

הכי נוח יהיה שהשורה השניה תהיה כולה 0 והשלישית כולה 1, ובשורה הרביעית כמה שיותר 1-ים כדי שנקבלים מלבנים של רביעיות במקום זוגות. כך נקבל סה"כ את הכיסוי הבא



שנותן לנו את הביטוי $wx+x'y+x'z$. זה לא ביטוי מינימלי כי אפשר לבחור שכל הצירופים האדישים יהיו 1 חוץ מ-$(1,1,1,0)$

ואז נקבל את הכיסוי הבא עם אותו מספר ליטרלים



פונקציות שלמות

הגדרה קבוצה F של פ' בוליאניות נקראת שלמה אם ניתן לממש כל פ' בוליאנית בעזרת פ' בקבוצה.

משפט $\{+,\cdot,'\}$ היא שלמה.

הוכחה: כל פ' בוליאנית ניתנת להצגה כסכום מכפלות שדורש רק את שלושת הפעולות הללו. ■

מסקנה $\{\cdot,'\},\{+,'\}$ הן שלמות.

הוכחה: עם דה-מורגן אפשר לייצר $+$ עם $\cdot,'$ ול-$\cdot$ עם $+,'$. ■

דוגמה NAND, כלומר $(x\cdot y)'$ היא פ' שלמה. ראשית ניתן להשיג NOT ($'$) באמצעות $x'=(x\cdot x)'$ ו-AND ($\cdot$) אפשר להשיג באמצעות NOT על NAND, ששניהם כבר יש לנו.

דוגמה הוכיחו כי $f(x,y,z)=x'+yz$ והפ' הקבועות $0,1$ הן קבוצה שלמה.

ראשית ל-NOT ניתן להגיע באמצעות $f(x,0,0)=x'+0\cdot 0=x'$. ל-AND אפשר להגיע ע"י $f(1,x,y)$. לכן ניתן לייצר קבוצה שלמה כלומר הקבוצה המקורית היא שלמה. אפשר גם להגיע ל-OR ע"י $f(f(x,0,0),y,1)=(x')'+y\cdot 1=x+y$

למעשה לא צריך את שני הקבועים כי ברגע שיש NOT עם אחד הקבועים אפשר להגיע לקבוע אחר עם NOT על הקבוע שכן יש לנו.

שבוע $\mathbb{III}$ ו

הרצאה

דוגמה נתונה הפ' עם טבלת האמת הבאה

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

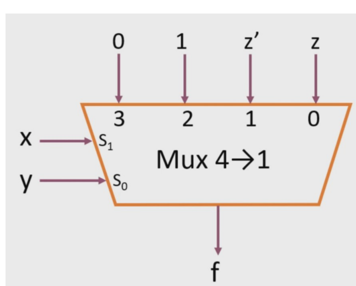
- נממש את הפ' עם $\text{MUX}\ 1 \to 8$ יחיד.

כל מה שצריך לעשות זה לחבר את x, y, z ל-s_0, s_1, s_2 בהתאמה ולהציב בקלטים $x_0, \ldots, x_7$ של ה-MUX את ערכי f בטבלת האמת לפי הסדר. כך נבחר עבור כל צירוף (x, y, z) בדיוק את התוצאה מתוך ערכי טבלת האמת של f.

- עתה נממש עם $\text{MUX}\ 1 \to 4$ יחיד ועוד שער אחד כלשהו. כאן צריך יותר להתאמץ. ראשית נביט בטבלה כאשר (x, y) מופרדים מ-z, כך שלכל (x, y) יש שני צירופים עם z עם ערכים אפשריים.

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

נבנה MUX יחיד עם סלקטורים x, y ובקלט ה-i נשים או קבוע אם שני הצירופים עם z נותנים את אותו הערך, או z או z' בהתאם לערך שהערך משתנה (שכנעו עצמכם שאכן אלו כל האפשרויות). כך נקבל את השער הבא



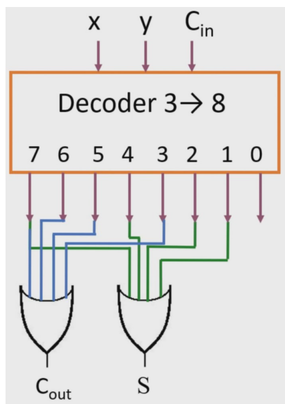
אנחנו מקיימים את הדרישות כי יש MUX אחד ושער אחד שהוא NOT על הקלט השני ל-MUX.

דוגמה נממש FA (שמקבל x, y, C_{in} ופולט S, C_{out} כזכור) שיש לו את טבלת האמת הבאה (לצורך נוחות) עם:

x	y	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- מפענח $3 \to 8$ ושערי OR.

נבחר קלטים למפענח x, y, C_{in}, ואז על הפלטים נצמיד שער OR אחד לפלט S ואחד לפלט C_{out}. סה"כ זה יראה כך (רק $(1,1,1)$ מוציא חוט לשני ה-OR-ים, בהתאם לטבלת האמת)

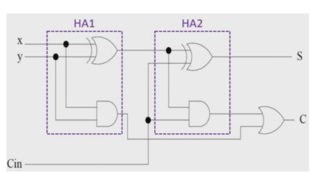


- MUX-ים $1 \to 8$.

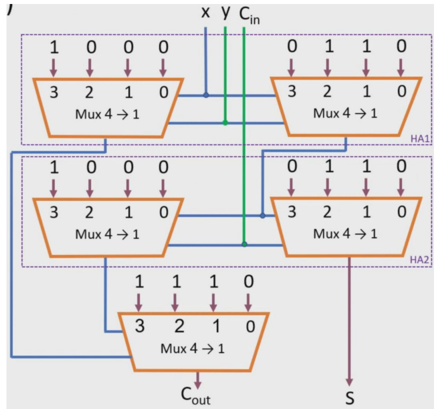
נבחר את הסלקטורים להיות x, y, C_{in} ונשבץ בשמונת הקלטים של ה-MUX את ערכי S בטבלת האמת לכל צירוף של הסלקטורים (הקלטים), ואותו הדבר שוב עם C_{out}.

- MUX-ים $1 \to 4$.

זה כבר יותר מורכב, ודורש פירוק של FA לשני Half Adder-ים שלא הזכרנו כאן, אבל הם שערים שמקבלים x, y ופולטים S, C_{out}. מימוש די פשוט ניתן לראות בתוך המלבן HA1, הסכום הוא XOR הקלטים והנשא הוא AND על הקלטים.



כך נוכל לחבר יחד שנים כאלו כדי לחשב $x + y + C_{in} = (x+y) + C_{in}$. HA הוא מעגל עם שני קלטים ולכן קל לממש אותו עם MUX $1 \to 4$ (משבצים את ערכי טבלת האמת כאמור) וכל שנותר הוא להרכיב שני HA לאחד יותר גדול (ראו איור)

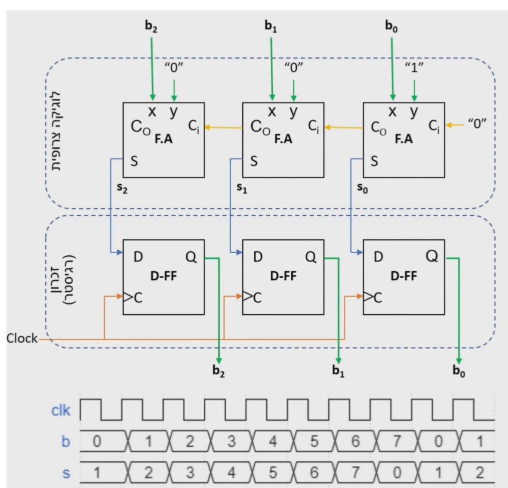


לאחר ראינו DFF, שמקבל קלט D ושעון ומשנה את פלטו בעת עליית השעון לערך של D (זו וריאציית ה-Positive Edge) נוכל על בסיס יחידה זו לבנות יחידות יותר מורכבות.

דוגמה Toggle FF מקבל T ושעון ומחשב בכל עליית שעון $Q\left(t+1\right)\ =\ \text{XOR}\left(T,Q\left(t\right)\right)$ ע"י חיווט הפלט Q יחד עם T אל תוך XOR שמוזן ל-D (קלט ה-DFF הפנימי).

דוגמה JK-FF מקבל J,K ושעון ומחשב בכל עליית שעון $Q\left(t+1\right)=JQ'\left(t\right)+K'Q\left(t\right)$ ע"י הרכבת J,K והפלט Q בשער לוגי שתוצאתו מוזנת ל-D של ה-DFF הפנימי.

דוגמה נממש ספרן, שסופר את מספר עליות השעון.



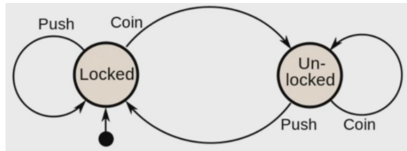
הרעיון כאן הוא פשוט אבל המימוש לא כל כך: בכל עליית שעון, ה-FA הימני ביותר מכניס עוד ערך 1 לסכום שמוחזק ע"י כל המעגל. הסכום מפועפע בכל עליית שעון ל-DFF הבא (וה-FA המתאים לו), וכך ה-DFF-ים מחזיקים שלושה ביטים שמייצגים מספר שערכו עולה באחד בכל עליית שעון (הסטודנטית המשקיעה תריץ את שלושת המחזורים בראש/על נייר ותראה שזה אכן עובד).

Finite State Machine

הגדרה FSM הוא מודל חישובי עם מספר סופי של מצבים, מצב התחלתי, מספר סופי של קלטים ופלטים, פ' מעברים $\{\text{מצבים}\}\times\{\text{קלטים}\}\rightarrow$ $\{\text{מצבים}\}$ ופ' פלט.

דוגמה שער מסתובב (Turnstile) יכול להיות פתוח או סגור, אם הוא מקבל מטבע והוא נעול, הוא נפתח, אם הוא לא נעול ונדחף, הוא ננעל וכו'. כן המצבים הם "פתוח" ו"נעול", הקלטים הם מטבע ודחיפה, הפלט עבור "פתוח" הוא "אפשר לעבור" ובהתאמה עבור "נעול".

נוכל לייצג את ה-FSM עם גרף

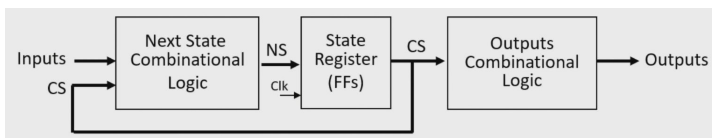


או טבלה (יש כמה דרכים)

State	Output	Inputs	
		Coin	Push
Locked (Init)	Closed pass	Unlocked	Locked
Unlocked	Open pass	Unlocked	Locked

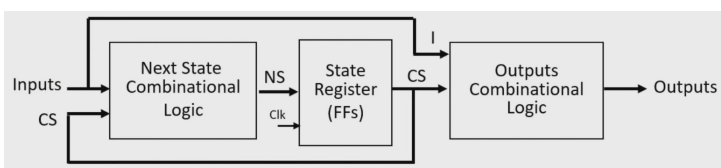
State	Output	Input	Next State
Locked (init)	Closed-pass	Coin	Unlocked
		Push	Locked
Unlocked	Open-pass	Coin	Unlocked
		Push	Locked

דוגמה מכונת Moore היא מכונה כבאיור (NS ו-CS הם המצב הבא והנוכחי בהתאמה), שמה שמייחד אותה הוא שהפלטים תלויים אך ורק במצב הנוכחי.



נשים לב שהרגיסטר שמורכב מ-FF-ים מחזיק את המצב הנוכחי, וערכו מחווט חזרה פנימה לחישוב המצב הבא שיכנס לרגיסטר במחזור הבא.

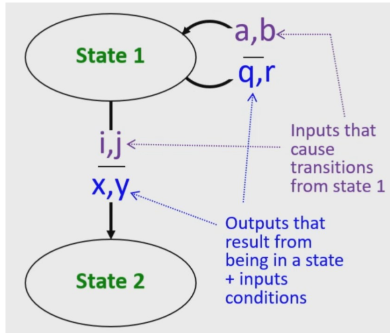
דוגמה מכונת Mealy היא מכונה שבה הפלטים תלויים גם במצב וגם בקלטים, והארכיטקטורה שלו היא כבאיור



הערה את הייצוג של הגרפי של מכונת Moore מציירים כמו אוטומט רגיל, רק שש כל מצב (מעגל) מכיל גם את הפלטים שהוא משרה.

את הייצוג הגרפי של מכונת Mealy מציירים כמו אוטומט רגיל, רק שעל החצים (המעברים) נוסיף את הפלטים שהקלטים על החץ יחד עם המצב שעוברים אליו משרים (ראו איור).

הערה הפלטים יושפעו מהקלט מהר יותר ב-Mealy כי הם מחוברים ישירות לפ' הפלטים, בעוד ב-Moore נדרשת עליית שעון כדי שישתנה המצב שמשרה פלט.



הערה אפשר לכתוב מכונות Mealy ששקולות למכונות Moore עם פחות מצבים, אבל החסרון הוא ש-Mealy גורם לבעיות עם תזמונים (שנלמד בהמשך).

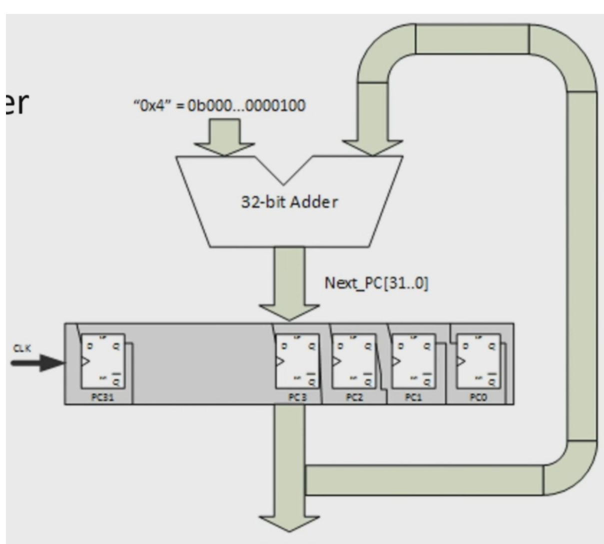
דוגמה נחזור לדוגמת השער המסתובב. נגדיר 0 השער נעול ו-1 הוא פתוח. לכן המעגל של פ' הפלט הוא חוט ישיר מהמצב לפלט כי הפלט זהה למצב. את המעברים ניתן לראות בטבלת האמת הבאה

Current State	Coin	Push	Next State
0	0	X	0
0	1	X	1
1	X	0	1
1	X	1	0

והמימוש של המעגל למעבר למצב הבא הוא די פשוט: אם Q הוא המצב הנוכחי ו-D המצב הבא (הקלט ל-DFF) אז $D = \text{Coin} \cdot Q' + \text{Push}' \cdot Q$ שזה ניתן למימוש עם שערים מאוד פשוטים.

דוגמה Program Counter נותן למעבד בכל פעם את הכתובת בזיכרון ממנה צריך לקרוא את הפקודה הבאה. הספרן פולט כתובת באורך 32 ביטים שקופצת בקפיצות של 4 (אלא אם הייתה קפיצה למיקום אחר) בגלל שכל מילה היא באורך 32 ביטים (בית הוא 8 ביט).

המימוש הוא די פשוט: נחזיק 32 DFF-ים שיחזיקו את הכתובת, נחווט ישירות את ה-DFF ל-32 הפלטים והמעגל לחישוב המצב הבא הוא FA עם $x = 4$ ו-$y = Q$ (Q המצב הנוכחי), או באיור ברזולוציה נמוכה משום מה זה יראה כך



זוהי מכונת Moore, שערכה מתעדכן פעם אחת בכל מחזור.

תזמון מעבד

הגדרה התדר של מעבד הוא מספר המחזורים של השעון בשנייה (ביחידות Hz).

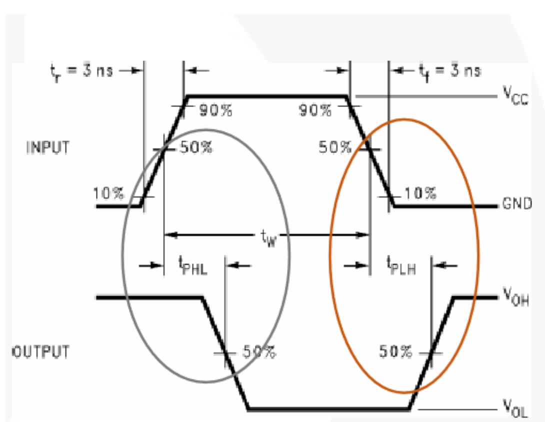
טעינה ופריקה של מטען לוקחים זמן ולכן מעבר או חסימת מעבר של זרם בתוך טרנזיסטור אינם מיידים (אלא מאוד מהירים). פרק הזמן הזה נקרא Propogation Delay.

פרמטרים של זמן פעפוע

- זמן הפעפוע מנמוך לגבוה (t_{PLH}): זמן הפעפוע כשהפלט עובר מנמוך (0) לגבוה (1). מודדים אותו החל משינוי ניכר בקלט ועד לעליית הפלט ל-50% מתח. בפועל מתח נחשב גבוה (ומתפרש כ-1) רק כשהוא 90% ומעלה מערכו המקסימלי, וכאן יש הנחה סמויה שהעליה והירידה של המתח הם מאוד מהירים ולכן מ-50% ל-90% אין הבדל משמעותי.
- זמן הפעפוע מגבוה לנמוך (t_{PHL}): זמן הפעפוע כשהפלט עובר מגבוה לנמוך, מחושב ע"י הפרש הזמנים בין שינוי ניכר בקלט ועד לירידת הפלט למתחת ל-50%.
- זמן עלייה (t_r, **R**ise): הזמן שלוקח לעלות מ-10% מתח ל-90% מתח.
- זמן ירידה (t_f, **F**all): הזמן שלוקח לרדת מ-90% ל-10% מתח.
- זמן פעפוע (t_{pd}): כש-$t_{\text{PHL}} = t_{\text{PLH}}$, נקרא להם t_{pd}.

הערה t_r, t_f הם מאוד קטנים (ננו-שניות).

דוגמה בשער NOT כלשהו מקבלים את הגרף הבא של הפלט כתלות בקלט.

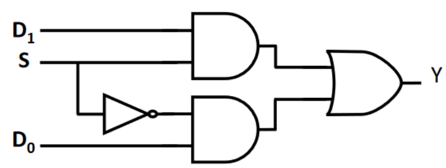


נשים לב שהשינוי הוא לא מיידי. במקרה הזה השינוי הניכר בקלט הוא חצייתו (מלמעלה או מלמטה) של הקלט את רף 50% המתח.

הגדרה עבור t_{pd} יש ערך טיפוסי, ערך מקסימלי (t_{pd-max}) שאומר אחרי כמה זמן בטוח הפלטים כבר ישתנו וערך מינימלי (t_{pd-min}) שמבטיח מתחת לאיזה רף בטוח הערכים לא ישתנו.

הערה $t_{pd-min/max}$ יכולים להיות שונים עבור שינוי בקלטים שונים (קלט x משפיע יותר מהר מ-y).

דוגמה נתון השער MUX שממומש באופן הבא



וחסמי זמן פעפוע לשערים

t_{pd-max}	t_{pd-min}	שער
$5ns$	$2ns$	NOT
$8ns$	$4ns$	AND
$10ns$	$5ns$	OR

- מהו t_{pd-max} של השער כולו?

עבור ההשפעה $Y \to D_1$ (כמה זמן לאחר שינוי D_1 Y ישתנה) נצטרך לעבור דרך AND ו-OR כלומר סה"כ $18ns$, וכך גם עבור D_0.

עבור $Y \to S$ זמן הפעפוע המקסימלי הוא $\max\left(\text{AND}+\text{OR}, \text{NOT}+\text{AND}+\text{OR}\right)=\max\left(18,23\right)=23$.

זמן הפעפוע של השער כולו הוא המקסימום של כל המסלולים, כלומר $23ns$.

- מהו t_{pd-min} של השער כולו?

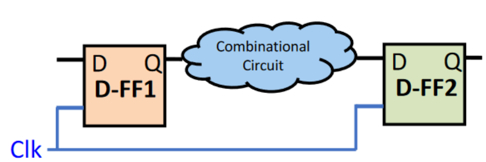
עתה נבחר את המסלול הקצר ביותר, שהוא כמובן $D_0 \to Y$ (או $D_1 \to Y$) שדורש $4ns+5ns=9ns$, וזהו t_{pd-min}.

הגדרה זמן הפעפוע של D ל-Q מחושב (מוגדר) החל מעלייה של השעון ל-50% ועד לשינוי Q והוא נקרא t_v (או t_{PCQ}) והוא למעשה מגדיר אחרי כמה זמן לאחר עליית השעון נוכל להחשיב את הקלט כחוקי. t_{v-min} מבטיח עד מתי Q ישאר בערכו הקודם ו-t_{v-max} מבטיח החל ממתי יהיה הערך החדש.

הגדרה כדי ש-Q יהיה תקין, D צריך להיות יציב ולא להשתנות למשך פרק זמן לפני עליית השעון, זהו t_s (**S**etup), וגם לאחר עליית השעון, זהו t_h (**H**old).

הערה $t_s>0$ כי בתוך ה-DFF ה-Master משנה את ערכו קצת לפני ה-Slave ולכן הערך שם צריך לא להשתנות כדי של-Slave יהיה את הערך הנכון.

דוגמה נביט בקונסטרוקציה הבאה



נניח שזמן מחזור השעון הוא t_{cyc} (הזמן בין עליית שעון אחת לשניה), לכן קצב השעון הוא $f=\frac{1}{t_{cyc}}$. נניח כי זמן הפעפוע המקסימלי של השער הוא t_{pd-max}. מהו זמן המחזור המינימלי כדי שהמעגל יהיה תקין, כלומר כדי שהתוצאה תגיע מהקלט ל-DFF1 עד לפלט של DFF2 תוך שני מחזורים (בראשון הקלטים עוברים את השער ובשני הם כבר מופיעים בצד השני)?

- זמן המחזור צריך לקיים את הדרישה

$$t_{cyc} \geq t_{v-max} + t_{pd-max} + t_{setup}$$

כי צריך קודם לחכות שהפלט של DFF1 יהיה חוקי (t_{v-max}), אז לתת לו לעבור את כל השער (t_{pd-max}) ואז שהפלט יהיה יציב מספיק זמן לפני עליית השעון הבאה כדי שיעבור בהצלחה ל-Q של 2DFF לאחר העליה.

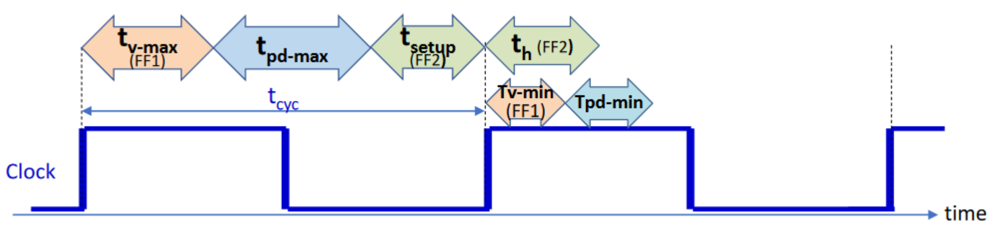
- מה צריך להיות t_h כדי שהשער יהיה תקין?

בפעם השנייה שבה ישתנה הערך, נצטרך ש-t_h של DFF2 יהיה פחות מהזמן שלוקח ל-D של DFF2 להשתנות בהשפעת ה-Q החדש של DFF1. כלומר, חייב להתקיים

$$t_h \leq t_{v-min} + t_{pd-min}$$

כי הזמן שלוקח לקלט לעבור מ-$Q\ (DFF1)$ ל-$D\ (DFF2)$ חסום מלמטה ע"י הזמן המינימלי שעבורו $Q\ (DFF1)$ לא ישתנה לאחר עליית השעון (t_{v-min}) ועוד הזמן המינימלי שעבורו $D\ (DFF2)$ לא יקבל ערך חדש כפלט של המעגל (t_{pd-min}).

סה"כ מהלך התזמונים כפ' של השעון הוא באיור



הערה אם אין לנו דרך לשלוט ב-t_h ונרצה עדיין מעגל חוקי, אפשר לחייב את t_{pd-min} להיות יותר גדול ע"י הוספת שני NOT-ים למסלול הקצר ביותר במעגל (הוא לרוב לא הארוך ביותר) וכך לא להשפיע על התוצאות אבל כן על התזמון המינימלי.

דוגמה נתונה הקונסטרוקציה הבאה עם MUX, שלו (הכל ביחידות ns) $t_{pd-min}=9, t_{pd-max}=23$ ו-DFF שלו $t_{v-min}=2, t_{v-max}=7, t_s=3, t_h=5$.

- מהו זמן המחזור המינימלי האפשרי?

כדי שהמעגל יהיה תקין, צריך שמסלול הפעפוע הארוך ביותר במבנה יהיה כולו מוכל במחזור אחד. המסלול הארוך ביותר הוא משינוי ב-$Q\ (/S0)$, דרך חישוב ה-MUX ועד לשינוי הערך ב-D, כשצריך לקחת בחשבון שלפני תחילת המחזור הבא נצטרך ש-D יהיה יציב ל-t_{setup} ננו-שניות. סה"כ נצטרך שיתקיים (בדומה לדוגמה הקודמת)

$$t_{cyc} \geq t_{v(max)} + t_{pd(max)} + t_{setup} = 7 + 23 + 3 = 33$$

* מהי הדרישה על t_h כדי לקבל מבנה תקין?

נצטרך שהערך של D לא ישתנה מוקדם מדי לאחר עליית השעון, בפרט שזמן שינוי הערך Q וחישוב ה-MUX יקחו יותר מאשר t_h, כלומר $t_h \le t_{v(min)} + t_{pd(min)} = 2 + 9$ ואכן הנתונים מקיימים את הנדרש ($t_h = 5 \le 9$).

סה"כ $T_{cyc(min)} = 33ns$ כלומר $F_{max} = \frac{1}{33ns} = 30MHz$.

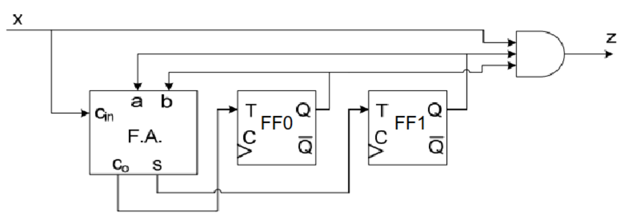
עד כה התעלמנו מהקלט האמיתי של המעגל שמחובר ל-$S1$. הוא עצמו גם חייב לקיים דרישה על השינוי שלו ביחס לשעון וכך נוכל להתייחס אליו כסיגנל שמגיע מפלט אחר והניתוח יהיה כנ"ל.

כל קלט למעגל אחד הוא פלט של מעגל אחר, ולכן יכולים להיות להם ערכי t_v, t_{v-min} שמושרים ממעגל אחר.

כל פלט למעגל כלשהו הוא קלט למעגל אחר ולכן הפלטים צריכים לענות על דרישות t_h, t_s מסוימות גם כן.

הערה כשננתח תזמון של מעגל, נסתכל על כל המסלולים שמתחילים בכניסה ל-DFF ונגמ

דוגמה נסתכל על המבנה הבא, כשנתון $t^x_{v(max)} = 15, t^x_{v(min)} = 0$ והדרישות על הפלט $t_{setup} = 9, t_{hold} = 0$ (z צריך להיות יציב למשך $9ns$ לפני עליית השעון, ו-$0ns$ אחרי)



עם הנתונים

Parameter	t_{pd-max}	t_{pd-min}	t_{setup}	t_{hold}	t_v	t_{v-min}
AND 3 inputs	3	1				
FA {a,b,Cin}->S	12	3				
FA {a,b,Cin}->Co	8	3				
T-FF			7	2	4	0

* מהו $T_{cyc-min}$? ננתח את כל המסלולים שנגמרים בפלט (כי לפלט יש דרישות ביחס לשעון, בפרט ל-DFF יש t_{setup} טכני ולפלט יש כזה שנדרש מאיתנו)

  – מסלולים שנגמרים ב-z: המסלול המקסימלי הוא באורך

$$t_{setup} + t^{AND}_{pd} + \max\left\{t^{FF1}_v, t^{FF0}_v, t^x_v\right\} = 9 + 3 + \max\left\{4, 4, 15\right\} = 27ns$$

כי z צריך להיות יציב t_{setup} זמן לפני עליית השעון, ערכו מחושב ע"י AND אז מוסיפים את זמן הפעפוע דרכו, וקלט ה-AND הם פלטי FF1, FF0 ו-x בהתאמה, שערכם מתעדכן למחזור הנוכחי לאחר ה-t_v של כל אחד מהם (כאן אנחנו מסמנים $t_v = t_{v(max)}$).

– מסלולים שנגמרים ב-$T\left(FF1\right)$: המסלול המקסימלי הוא באורך

$$t_{setup}^{FF1}+t_{pd(\to S)}^{FA}+\max\left\{t_{v}^{FF1},t_{v}^{FF0},x_{valid}\right\}=7+12+\max\left\{4,4,15\right\}=34ns$$

– מסלולים שנגמרים ב-$T\left(FF0\right)$:

$$t_{setup}^{FF0}+t_{pd(\to C_{0})}^{FA}+\max\left\{t_{v}^{FF1},t_{v}^{FF0},x_{valid}\right\}=7+8+\max\left\{4,4,15\right\}=30ns$$

ולכן סה"כ נצטרך $T_{cyc-min}\geq\max\left\{27,34,30\right\}=34ns$.

• האם מתקיימת הדרישה על ה-Min Delay?

כן! עבור z מתקיים $t_{h}^{z}=0ns$ והזמן המינימלי לפעפוע הוא

$$t_{pd(min)}^{AND}+\max\left\{t_{v(min)}^{x},t_{v(min)}^{FF}\right\}=1+\min\left\{0,0\right\}=1ns$$

כלומר מתקיימת הדרישה ועבור FF0, FF1 יש $t_{h}^{FF}=2ns$ והזמן המינימלי לפעפוע הוא

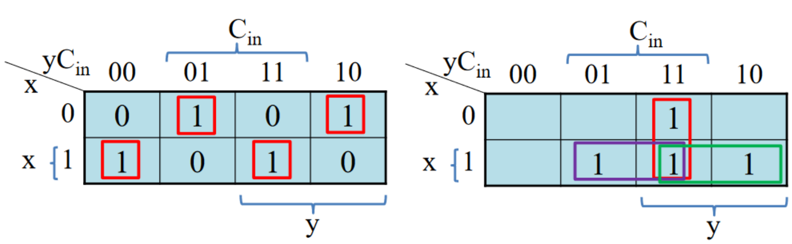
$$t_{fpd(min)}^{FA}+\min\left\{t_{v(min)}^{FF},t_{v(min)}^{x}\right\}=3ns$$

(כי הערך החדש צריך לעבור דרך ה-FA ולהגיע או משינוי ב-x או משינוי ב-Q של אחד ה-FF-ים) ולכן מתקיימת הדרישה גם כן.

תרגול

הגדרה מעגל צירופי (קומבינטורי) הוא מעגל שיש לו כניסות ויציאות כאשר האחרונות תלויות בכל ערך רגעי של הראשונות. מעגל סדרתי הוא מעגל שפלטיו תלויים גם ביחידת זכרון שמחוברת לשעון.

דוגמה כיצד נממש Full Adder (כזכור קלטים x,y,C_{in} ופלטים S,C_{out})? ראשית נמלא את טבלת הקרנו לפלט S (משמאל) ו-C_{out} (מימין)



ולכן אפשר לממש את S עם OR על ארבעה פלטי AND (שכוללים קלטים שעוברים דרך מהפך) ואת C_{out} אפשר קצת יותר יעיל. ראינו בהרצאה שהמימוש של FA כולל בתוכו שני HA.

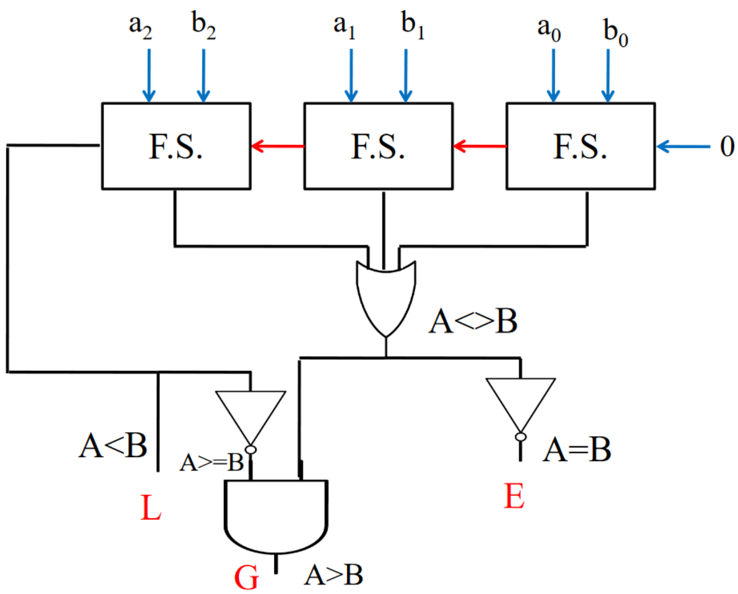
הגדרה מחסרים מחשבים חיסור ספרות בינאריות. עתן נפלוט את ההפרש (**בערך מוחלט**) ו-Borrow Out שיגיד לנו כמה נותר לחסר מעבר להפרש. הסטודנטית המשקיעה תממש חצי-מחסר ומחסר מלא.

הגדרה משווים הם מעגלים שבודקים איזה קלט יותר גדול.

- דרך אחת לממש זאת היא באמצעות השוואה מה-MSB ל-LSB כאשר בפעם הראשונה שיש אי-שוויון בביטים נבחר את האחד שקלטו גדול יותר (1 לעומת 0).
- דרך אחרת היא באמצעות מחסרים: $A > B \iff A - B > 0$ וכו'.

דוגמה נתונים הקלטים $A = a_2a_1a_0, B = b_2b_1b_0$ ממשו עם מחסרים מעגל שפלט ביטים G, E, L שערך כל אחד מהם 1 אם"ם $A > B, A = B, A < E$ בהתאמה.

נממש את המעגל כבאיור



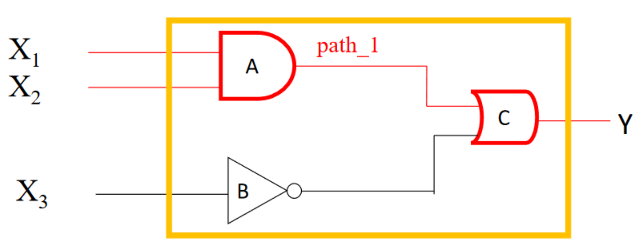
אם $A < B$ בהכרח ישאר לנו Borrow Out כי אם $a_2 = b_2$ אז $a_1a_0 < b_1b_0$ ואז יהיה BO מלפני שימשיך הלאה ואחרת $a_2 < b_2$ ולכן בוודאי שיהיה BO.

אם כל הביטים זהים נצטרך NOR על כל הפרשים (כל החיסורים פולטים 0) ואכן זה מה שבנינו.

בשיטת האלימנציה, G הוא 1 אם $A \neq B$ וגם $A \geq B$, כלומר אם ה-OR הוא 1 וגם ה-L הוא 0 וכך אכן מופיע במעגל.

הערה השימוש ב-10% ו-90% כרף לחישובי תזמון נובע מכך שקשה לאפיין את פריקת הקבל כתהליך לינארי בקצוות, ולכן מתעלמים מהם.

דוגמה נביט בפ' $X_1 * X_2 + X_3'$ שממומשת באופן הבא



מתקיים $t_{pd}\left(A\right)=\max\left\{ t_{\text{PLH}}\left(A\right),t_{\text{PHL}}\left(A\right)\right\}$ ובדומה עבור C. יש לנו שני מסלולים במעגל, הראשון מ-X_1, X_2 ל-Y והשני מ-X_3 ל-Y.

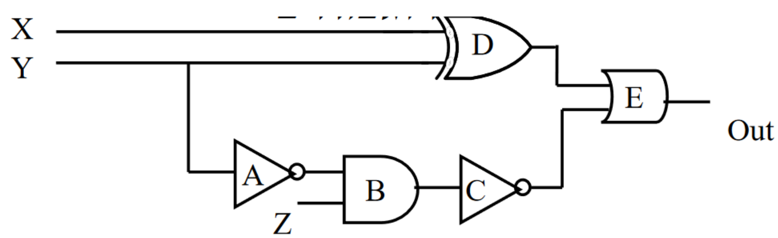
לכן $t_{pd}\left(\text{מסלול ראשון}\right)=t_{pd}\left(A\right)+t_{pd}\left(C\right)$ ובהתאמה $t_{pd}\left(\text{מסלול שני}\right)=t_{pd}\left(B\right)+t_{pd}\left(C\right)$ תחת הנתונים הבאים (כאשר $t_{cd}=t_{tp-min}$ מלשון Contamination)

Data in ns	X_2	A	B	C
t_{PHL}	-	100	**90**	80
t_{PLH}	-	**110**	70	**100**
t_{cd}	-	12	8	10
t_r	**14**	20	12	**18**
t_f	15	17	13	19

נחשב את ה-t_{pd} של המעגל כולו. עבור כל שער, $t_{pd}=\max\left\{ t_{\text{PHL}},t_{\text{PLH}}\right\}$ ולכן $t_{pd}^{p1}=110+100=210ns$ וכן $t_{pd}^{p2}=90+100=190ns$.

נחשב את ה-$t_{pd(min)}$ של המעגל. $t_{pd(min)}^{p1}=12+10=22ns$ וכן $t_{pd(min)}^{p2}=8+10=18ns$.

דוגמה נתונים המעגל והנתונים הבאים



Data in ns	A	B	C	D	E
t_{pd}	15	25	15	60	20
t_{pd-min}	5	5	5	10	5

המסלולים של שערים מהקלטים לפלטים הם $D\to E$, $A\to B\to C\to E$ ו-$B\to C\to E$ שלהם t_{pd} 80, 75 ו-60 בהתאמה ו-$t_{pd(min)}$ 15, 20 ו-15 בהתאמה.

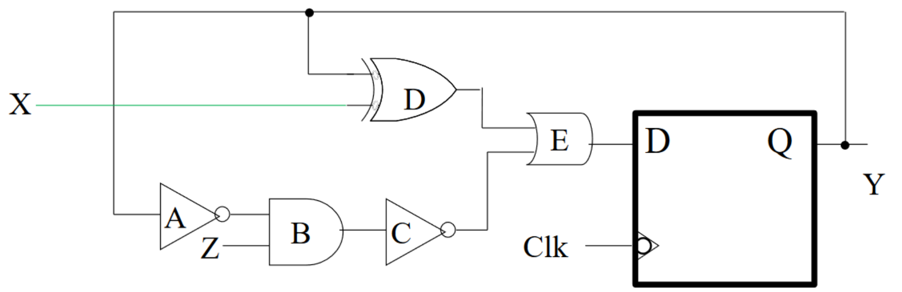
לכן $t_{pd}=80$ ו-$t_{pd(min)}=15$.

הערה כל עוד אין מעגלים סדרתיים, חישוב t_{pd} נעשה ע"י מקסימום הזמנים על כל המסלולים מפלטים לקלטים ו-$t_{pd(min)}$ ע"י מינימום.

הערה עבור כל מעגל סדרתי חייב להתקיים $t_{hold}\le t_{v(min)}^{FF}+t_{pd(min)}^{\text{לוגיקה}}$ כדי שהשינוי הכי מהיר במעגל יקח יותר מאשר הזמן שהפלט צריך להישאר זהה אחרי עליית השעון.

הערה הקלט למעגל חייב להתייצב לאחר לכל היותר $t_{setup}+t_{pd}^{\text{לוגיקה}}$ זמן כדי שה-FF יוכל לחשב באופן תקין את Q.

דוגמה נתון המעגל הסדרתי הבא עם הנתונים תחתיו



Data in ns	A	B	C	D	E
t_{pd}	15	25	15	60	20
$t_{pd\text{-}min}$	5	5	5	10	5

כדי להשלים את הניתוח של תזמון המעגל נצטרך את האילוצים על ה-DFF $t_v = 10ns$, $t_{v(min)} = 5ns$, $t_{setup} = 10ns$, $t_{hold} = 25ns$.

- האם המעגל תקין?

לא! חייב להתקיים $20 = 5 + 10 + 5 = 5 + t_{pd(min)}^{D\to E} = 5 + t_{pd(min)} \leq t_{hold} = 25$ סתירה.

- כיצד נפתור את הבעיה?

נוסיף דיליי על המסלול הקצר ביותר, בפרט נוסיף שני NOT-ים על החוט בין Q לקלט העליון של D (ושל A). עכשיו המסלול הקצר ביותר יש לו $t_{pd(min)} = 25$ ועכשיו נקבל $30 = 5 + 25 \leq t_{hold} = 25$ כלומר זה כן תקין. החסרון הוא שהתדר המקסימלי האפשרי ירד כי זמן המחזור המינימלי עלה כתוצאה מהוספת שני ה-NOT-ים.

שבוע $\mathbb{IV}$ | MIPS

הרצאה

הגדרה Instruction Set Architecture היא אוסף כללים שמתכנת צריך לענות להם כשהוא מפתח למעבד.

דוגמה אנחנו נלמד על MIPS, אבל במציאות פופולריים מאוד x86 ו-ARM, וגם RISC-V שהוא פרויקט קוד פתוח.

הגדרה מיקרו-ארכיטקטורה היא מימוש של ISA.

דוגמה המימוש של אינטל ו-AMD ל-x86 הוא מיקרו-ארכיטקטורה.

לצורך האבסטרקציה שלנו, מעבד הוא מכונת מצבים המחוברת לזיכרון, כאשר המצב כולל רגיסטרים שערכם משתנה על ידי פקודות. הזיכרון הוא מערך שניגשים אליו לפי אינדקס (בית אחד בכל פעם). כל מעבד מריץ את התכנית הבאה:

1. קרא את הפקודה הבאה מהזיכרון בכתובת שברגיסטר PC (Program Counter).

2. הוסף לרגיסטר PC את מספר הבתים שתופסת פקודה (התקדמות לפקודה הבאה).

3. בצע את הפקודה (ושנה את מצב המעבד).

4. חזור לשלב 1.

בהינתן תוכנה בשפה עילית (שמתקמפלת), נתרגם את הקוד לסדרת פקודות אסמבלי באמעות קומפיילר. לאחר מכן נשתמש באסמבלר כדי לתרגם את קוד האסמביל לבינארי, שאותו המעבר כבר יודע להריץ.

הערה לעתים הקוד שלנו ישתמש בספריות חיצוניות או בקבצי קוד אחרים, ועל איחוי כל הפקודות לקובץ אובייקט יחיד.

הפקודות באמסבלי הן פקודות שהמעבד יודע לבצע (ה-ISA של המעבד), אבל לפני שהן מקודדות ל-1-ים ו-0-ים עבור המעבד.

CISC vs RISC

- Complex Instruction Set Computer זו גישה לפיה פקודות האסמבלי יהיו קרובות ככל הניתן לשפה עילית, וכך להוריד את מספר הפקודות בתוכנה. בפועל זה ממומש ע"י פקודות שמפורקות למיקר-פעולות ע"י החומרה. ל-ISA הזה יש הרבה מאוד פקודות, בפורמטים שונים והרכבה של פקודות שונות אחת על השניה, ואפשר אפילו להריץ פקודות ישירות על הזיכרון שמסתירות את המעבר ברגיסטרים. אורך הפקודות יכול להשתנות, כאשר נקודד פקודות שכיחות באמצעות בית אחד, עד לפקודות הנדירות ביותר שהן באורך 15 בתים.

  x86 ו-DSP הולכים לפי גישת CISC.

- Reduced Instruction Set Computer היא גישה לפיה יש לשמור את מספר הפקודות מצומצם וכך לפשט את פעולות החומרה, ולתת לקומפיילר לעשות את העבודה הקשה של בחירת הפעולות ואופטימיזציה. הפקודות הן די פשוטות והן רק בין רגיסטרים (ולא על הזכרון), וצריך באופן מפורש לטעון ולשמור נתונים בזיכרון. אורך הפקודות הוא קבוע.

  MIPS, ARM ו-RISC-V הולכים לפי גישת RISC.

הערה המגמה עם הזמן היא לעבור מ-CISC ל-RISC משום שבעבר קשה היה לכתוב קומפיילרים יעילים ולכן נדרשה המורכבות של פקודות האסמבלי, ואילו עם חלוף הזמן נהיה קל ויעיל יותר לכתוב קומפיילרים (בשפה עילית) שיבצעו את הפעולה המורכבת בעצמם.

דוגמה עבור העתקה של 100 ערכים בין מערך אחד לאחד ב-c, יש ב-x86 פקודה אחת שמקבלת את מספר הבתים להעתקה והפוינטרים והחומרה כבר תממש את ההעתקה, לעומת RISC שם צריך לממש לולאה שמעתיקה לרגיסטר ואז לזיכרון מילה מילה.

MIPS

במעבד MIPS יש 32 רגיסטרים, $\$0, \ldots, \$31$, שכל אחד מהם בגודל 32 ביט, שמכונה "מילה" (4 בתים). מספר הרגיסטרים נקבע כך לאחר ניתוח של מספר המשתנים בתוכנות מדגמיות, כאשר אם יש יותר משתנים מרגיסטרים נשתמש בזיכרון הראשי כדי להחזיק את ערכם. לחלק מהרגיסטרים יש יעודים ספציפיים, כפי שניתן לראות בטבלה הבאה

Name	Register Number	Usage	Preserve on call?
$zero	0	constant 0 (hardware)	n.a.
$at	1	reserved for assembler	n.a.
$v0 - $v1	2-3	returned values	no
$a0 - $a3	4-7	arguments	yes
$t0 - $t7	8-15	temporaries	no
$s0 - $s7	16-23	saved values	yes
$t8 - $t9	24-25	temporaries	no
$gp	28	global pointer	yes
$sp	29	stack pointer	yes
$fp	30	frame pointer	yes
$ra	31	return addr (hardware)	yes

נשים לב שרבים מהרגיסטרים משמשים פעילות תקינה של המחסנית (SP, FP, RA), חלק שומרים ארגומנטים וחלק משומשים לצרכים אחרים.

כל פקודה היא בגודל מילה (32 ביט), וכל פקודה מבצעת פעולה פשוטה, בין היתר פעולות אריתמטיות ולוגיות, גישה לזיכרון (load, store) וקפיצות ופקודות מותנות. הפקודות שמורות בזיכרון ובכל פעם נקרא מהזכרון את הפקודה ונריץ אותה.

פקודות אריתמטיות

- חיבור/חיסור: נביט בפקודה `add $d, $s, $t`. היא מחברת את $\$s, \$t$ ושמה את התוצאה ב-$\$d$ (כאשר שלושת הפרמטרים הם רגיסטרים), ובדומה `sub $d, $s, $t`

נשים לב שלא הודענו למעבד בשום מקום שאנחנו משתמשים בשיטת המשלים ל-2, כי אנחנו מניחים שמי שקימפל את הפקודה יודע מהקשר שהוא סוכם רגיסטרים שיש בהם כבר ערכים מיוצגים במשלים ל-2, ואם לא אז זה באג.

דוגמה נקמפל את הפקודה $f = (g + h) - (i + j);$

הקומפיילר יקצה רגיסטרים למשתנים ונקבל $\$s0 = (\$s1 + \$s2) - (\$s3 + \$s4)$, ואז לתרגום השורה לשפת אסמבלי יש כמה אפשרויות, הנאיבית ביותר מתוכן היא

```
add $t0, $s1, $s2

add $t1, $s3, $s4

sub $s0, $t0, $t1
```

אבל יש דרכים אחרות (לדוגמה לפתוח סוגריים). במתמטיקה אמנם התוצאות שקולות, אבל כאן יכול להיות שנקבל תוצאה אחרת בגלל overflow-ים.

- חיבור מידי: הפקודה `addi $t, $s, i` מחשבת $\$t = \$s + imm$ (כלומר חיבור בין רגיסטר וקבוע והשמה ברגיסטר).

כאן imm נתון לנו במשלים ל-2 אבל בגודל 16 ביט כדי שנוכל לכלול אותו בתוך קידוד הפקודה בגודל מילה אחת, ולכן נצטרך להרחיב אותו למשלים ל-2 בגודל 32 ביטים, פעולה זו נקראת Sign Extend, וניתן לממשה בקלות ע"י ריפוד בצד של ה-MSB עם ערך קבוע של 0 לחיוביים ו-1 לשליליים (למעשה ערך ה-MSB לפני הריפוד).

הערה לא צריך subi כי אפשר לממש בקלות עם addi כאשר ה-imm הוא מספר שלילי.

פעולות לוגיות

- bitwise AND: הפקודה היא `and $d, $s, $t` והיא מחשבת שער AND על כל שני ביטים מתאימים מ-$\$s$ ו-$\$t$ (בו זמנית על כל הביטים) ושומרת את התוצאה ב-$\$d$.

- הזזה לוגית: הפקודה `sll $d, $t, a` (מלשון Shift Left Logical) מזיזה a ביטים שמאלה (לכיוון ה-MSB) את הביטים של $\$t$ ושומרת את התוצאה ב-$\$d$.

מימין מכניסים אפסים ובנתיים מאבדים ביטים משמאל, כאשר במקרה שמספרים מיוצגים במשלים ל-2 זה יכול לאבד את ביט הסימן, ובכל מקרה נוכל לקבל overflow גם במקרה של טבעיים.

- הזזה אריתמטית: נשמור על ביט הסימן גם לאחר ההזזה במקום להתעלם ממנה. ראו טבלה שמשווה את כל ההזזות הקיימות ב-MIPS.

Instruction	Operation	Description
`sll $d, $t, a`	Shift Left Logical $d = $t << a	Shifts a register value left by the shift amount listed in the instruction and places the result in a third register. Zeroes are shifted in.
`sllv $d, $t, $s`	Shift Left Logical $d = $t << $s	Shifts a register value left by the value in a second register and places the result in a third register. Zeroes are shifted in.
`sra $d, $t, a`	Shift Right Arithmetic $d = $t >> a	Shifts a register value right by the shift amount (shamt) and places the value in the destination register. The sign bit is shifted in.
`srav $d, $t, $s`	Shift Right Arithmetic $d = $t >> $s	Shifts a register value right by the value in a second register and places the value in the destination register. The sign bit is shifted in.
`srl $d, $t, a`	Shift Right Logical $d = $t >>> a	Shifts a register value right by the shift amount (shamt) and places the value in the destination register. Zeroes are shifted in.
`srlv $d, $t, $s`	Shift Right Logical $d = $t >>> $s	Shifts a register value right by the amount specified in $s and places the value in the destination register. Zeroes are shifted in.

הזזה אריתמטית של ביט אחד שמאלה פרושה הכפלה ב-2 (עד כדי overflow) וימינה פרושה חלוקה ב-2 (עם עיגול כלפי מטה).

פעולות זכרון

הזכרון הוא מערך, שניגשים אליו באמצעות כתובות, כאשר כל כתובת מצביעה לבית אחד של מידע, למרות שבפועל ב-MIPS נקרא ונכתוב ביחידות של מילה (ארבעה בתים) מיושרות (כלומר כתובות שמתחלקות ב-4).

- קריאת מילה: `lw $t, i($s)` קוראת מילה מהזכרון בכתובת $s + imm (i הוא ה-immediate) ושומרת אותה ב-$t.

דוגמה `lw $t1, 0($a0)` קוראת מילה מהכתובת ששמורה ב-$a0 וכתובת אותו לרגיסטר $t.

- כתיבת מילה `sw $t, i($s)` כותבת את ארבעת הבתים ברגיסטר $t לכתובת $s + imm בזכרון.

דוגמה יש לנו מערך A עם שלושה ערכים (מספרים, בגודל ארבעה בתים), שכתובת הבסיס שלו שמורה ב-$s3. נוכל לגשת אל האיברים במערך באמצעות 0($s3), 4($s3), 8($s3).

דוגמה נניח שיש לנו את הפקודות ב-C

```
int A[100];

A[12] = h + A[8]
```

הקוד הזה יתורגם באסמבלי ל-

```
lw $t0, 32($s3) # load A[8] into a temporary register

add $t0, $s2, $t0 # add h to the temporary register

sw $t0, 48($s3) # save the result in A[12]
```

פילוסופיות ארגון זכרון

- ארכיטקטורת ואן-ניומן: זכרון אחד לפקודות התוכנה וגם למשתני התוכנה. כשקוראים מהזכרון, משמעות התוכן (פקודה או מידע) תלויה בפעולה שהובילה לקריאה. אפע"פ שתיתכן הפרדה פיזית בין החלקים, לוגית הם ממופים לאותו המקום.

  יתרונות אזור זכרון מאוחד, וקל לדבג תוכנות ולשנות את אופן הטעינה.

  חסרונות קריאת פקודות וקריאת מידע קורות על אותו המשאב.

  ממומש ב-x86, MIPS, ARM.

- ארכיטקטורת הארוורד: הזכרון של הקוד מופרד מהקוד של המידע, כך ש-lw קורא רק מידע ו-fetch לפקודות קורא רק פקודות.

  יתרונות אין גישה במקביל למשאבים שונים על אותו הפס - ביצועים יותר טובים.

  חסרונות חוסר גמישות בגודל הסגמנטים של קוד לעומת דאטא וקשה יותר לערוך את הקוד לדיבוג.

  ממומש בעיקר ב-DSP.

הערה גם בואן-ניומן המימוש בפועל יכול להיות באמצעות רכיבים פיזים שונים, הגישה תהיה באמצעות אותו מרחב כתובות (אמנם כתובות אחרות, אבל באותו מיפוי).

פקודות קפיצה והתניות

- קפיצה בלתי-מותנת: `j label` קופץ לאחר סיום הפקודה לשורת הקוד שמופיע לאחר ה-label (כפי שנכתב בקובץ ה-asm).

- קפיצה מותנת שוויון: `beq $s, $t, label` קפוץ ל-label אם $s==$t ואחרת ממשיכה לפקודה הבאה (ובדומה bne שקופצת אם $s!=$t).

דוגמה נתון הקוד הבא ב-C

```
if (i == j)
    f = g + h;
else
    f = g - h;
```

הוא יתורגם לקוד אסמבלי באופן הבא

```
    bne $s0, $s1, not_eq # s0=i, s1=j
    add $v0, $s2, $s4 # f = g+ h
    j cont
not_eq:
    sub $v0, $s2, $s4 # f = g - h
cont:
```

- השמה מותנית: `slt $d, $s, $t` משימה ב-$d 1 אם $t < $s ואחרת 0, בדומה `slti $t, $s, i` שם ב-$t 1 אם $s < imm ואחרת 0.

מימוש פונקציות באמסבלי

הגדרה הקוראת היא הפ' שקוראת לפ' אחרת, הנקראת היא הפ' שנקראת, הפרמטרים הם הערכים שמועברים מהקוראת לנקראת, התוצאות הם הערכים שהנקראת מחזירה לקוראת וכתובת החזרה היא הכתובת בזכרון הקוד של הפקודה שאחרי הקריאה לנקראת בקוד של הקוראת.

המחסנית היא אזור בזכרון שתוכנה יכולה להשתמש בו, והיא משמשת שמירה של מידע לוקאלית בעת הרצת פ' באופן שמאפשר קריאה מקוננת וחזרה מקריאות של פ'. מבנה הנתונים עובד בשיטת LIFO ואפשר או לדחוף (push) אלמנט לראש המחסנית, או להוציא (pop) את הערך העליון במחסנית.

הערה לעתים אפשר לגשת גם לערכים אקראיים במחסנית, ובכל מקרה ניגשים לכתובות ביחס לכתובת ראש המחסנית - Top of Stack, כאשר כל דבר מתחת אינו ואלידי. זאת משום שהמחסנית גדלה נגד כיוון הכתובות, כלומר הוספת ערך תזיז את TOS ארבעה בתים למטה.

שמירת רגיסטרים בעת קריאה וחזרה מפונקציה

הנקראת לא יודעת מה הקוראת עושה, ורק מצפה לפרמטרים נכונים ולהוציא תוצאות נכונות. לכן אם הקוראת משתמשת ברגיסטרים כלשהי, הנקראת תדרוס אותם בלי לדעת שהקוראת צריכה אותם. כדי לשמר את המצב לפני ואחרי קריאה לפ' יש calling convention; נניח ש-f (הקוראת) הייתה באמצע חישוב כשקראה ל-g (הנקראת). רק חלק מהרגיסטרים נשמרים, וע"י גורמים שונים:

- $t0-$t9 הם רגיסטרים זמניים ולכן באחריות f (הקוראת) לשמור אותם במקום אחר במהלך הקריאה ל-g.
- $s0-$s7 הם רגיסטרים סטטיים ולכן f מצפה שהם לא ישתנו, כך שאם g משתמשת בהם היא תצטרך לשחזר אותם לערכם בטרם קריאתה כדי ש-f תתפקד כמו שצריך.

- $v0-$v1-ו $a0-$a3 משמשים העברת והחזרת ארגומנטים ולכן הקוראת צריכה לשמר אותם אם היא רוצה להשתמש בערכים שהיא עצמה קיבלה (או תחזיר) בתור נקראת .
- $ra הוא הרגיסטר שמחזיק את כתובת החזרה מהפ', והיא נדרסת ע"י הפקודה jal בעת הקריאה לפ' הנקראת, כך שאחריות השימור היא על הקוראת.
- $sp, המצביע לראש המחסנית, שנדרש לכל הפ' על מנת לפעול באופן תקין, ולכן הנקראת נדרשת לשחזר אותו בעת סיום הקריאה לפ'.:

את הערכים נשמור תמיד במחסנית.

- כדי לדחוף (ולשמור) את $ra למחסנית נשתמש בפקודות

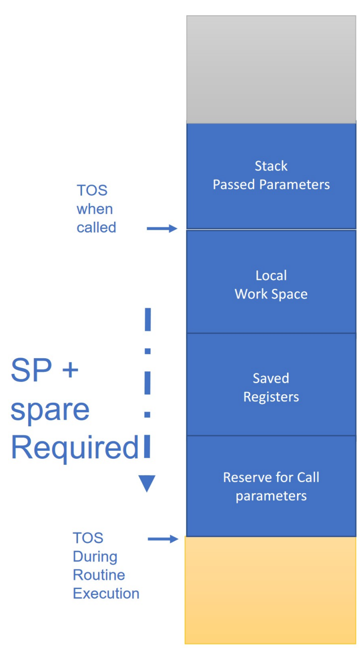
```
addi $sp, $sp, -4

sw $ra, ($sp)
```

כלומר קודם מזיזים את המצביע לראש המחסנית מילה אחת למטה בזיכרון (מעלים את גובה המחסנית) ואז כותבים לכתובת הבסיס של המחסנית את הערך של $ra, כך שהוא עכשיו בראש הערימה.

- כדי לגשת לערכים אפשר פשוט לבצע lw על כל היסט (חיובי) ביחס ל-$sp.
- כדי לעשות pop קודם נקרא את המילה ואז נזיז כלפי מעלה את $sp.

בעת קימפול נוכל לדעת כמה מקום פ' צריכה במחסנית (מבחינת משתנים מקומיים, שמירת רגיסטרים ומקום לקרוא לפ' אחרות). מבחינת סידור הזכרון בעת קריאה לפ', המחסנית תראה כך



פקודות קריאה לפונקציה

- `jal label` קופצת ל-label אבל לפניכן שמה ב-$ra את הערך הבא שיקבל PC אילולא קפצנו.

- `jr $s` קופצת לכתובת שברגיסטר $s.